A Comparison of Strict and Non-strict Semantics for Lists

Jerry R. Burch

Computer Science Department
California Institute of Technology

Caltech-CS-TR-88-12

A Comparison of Strict and Non-Strict Semantics for Lists

Jerry R. Burch

Master's Thesis

Caltech-CS-TR-88-12

This material is based upon work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings conclusions or recomendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

1. Introduction

Implementations of functional programming languages can be classified according to whether they apply *eager-evaluation* or *lazy-evaluation.* Eager-evaluation gives rise to *strict* semantics while lazy-evaluation gives rise to *non-strict* semantics. In this paper we define the syntax of a simple functional programming language, and specify strict and non-strict denotational semantics for that language. These semantics are specified by giving axioms for the domains and semantic functions involved. The axioms for the two different semantics are very similar, differing only in the specification of *cons*. However, this small difference results in the domains for the two semantics being quite different. Giving axioms, rather than just postulating particular domains and semantic functions, makes more explicit the similiarities of the strict and the non-strict semantics. We give a model of the axioms of the non-strict semantics in order to show their consistency, and show that any two such models are isomorphic.

It is important that a proposed semantics for a programming language does not make unrealistic demands on implementations of that language. To demonstrate that our non-strict semantics allow straightforward implementation, we present the kernel of an interpreter, and give a detailed sketch of a proof that this interpreter satisfies the non-strict semantics given by our axioms.

A formal semantics should also provide tools for proving properties of programs. Our semantics provide simple and powerful proof techniques that can be applied equally well to programs with strict and non-strict semantics. These proof techniques do not require the notion of admissible predicates [6]. We give example correctness proofs for programs under both semantics.

An explanation of some of our notation and terminology is needed. The symbol $\circ$ is used to designate (functional order) function composition, so that $(f \circ g)(x) = f(g(x))$. A relation $\sqsubseteq$ is a partial order relative to an equivalence relation $\equiv$ if it is reflexive ($x \sqsubseteq x$), antisymmetric ($x \sqsubseteq y$ and $y \sqsubseteq x$ implies $x \equiv y$), and transitive ($x \sqsubseteq y$ and $y \sqsubseteq z$ implies $x \sqsubseteq z$).

A sequence of objects $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq \ldots$ is called a chain. When we let $\{x_i\}$ be a chain, it is implicit that x_i exists for every integer $i \geq 0$ and that $x_i \sqsubseteq x_{i+1}$. We frequently have need for the *least upper bound* of a chain, which we write $\bigsqcup_{i=0}^{\infty} x_i$. The least upper bound is defined such that for all y

$$\forall i[x_i \sqsubseteq y] \iff \bigsqcup_{i=0}^{\infty} x_i \sqsubseteq y.$$

It is clear from this definition that if the least upper bound exists it is unique.

A function f of arity r is *monotonic* if $f(x_1, x_2, \ldots, x_r) \sqsubseteq f(y_1, y_2, \ldots, y_r)$ whenever $x_i \sqsubseteq y_i$ for all i between 1 and r. A function f of arity r is *continuous* if it is monotonic and

$$\bigsqcup_{j=0}^{\infty} f(x_{1,j}, x_{2,j}, \ldots, x_{r,j}) \equiv f(x_1, x_2, \ldots, x_r),$$

where for all i between 1 and r the chain $\{x_{i,j}\}$ has x_i as its least upper bound.

2. Syntax

In this section we describe the syntax of a simple functional programming language, L. The abstract syntax for terms in L is given by

$$\begin{aligned} E ::= {} & v \mid i \mid \mathit{true} \mid \mathit{false} \mid \Diamond \mid \mathcal{E} \mid \\ & E_1 : E_2 \mid \mathit{hd}(E_1) \mid \mathit{tl}(E_1) \mid \textbf{if } E_1 \textbf{ then } E_2 \textbf{ else } E_3 \mid \mathit{atom}(E_1) \mid \mathit{nil}(E_1) \mid \\ & E_1 + E_2 \mid E_1 - E_2 \mid E_1 \times E_2 \mid E_1 \textbf{ mod } E_2 \mid E_1 = E_2 \mid E_1 \leq E_2 \mid \\ & f(E_1, \ldots, E_r). \end{aligned}$$

The terms given by the first line of disjunctions are base terms. In these disjunctions we use v to designate that a term can be one of a countably infinite number of *object variables*, and we use i to designate that a term can be the decimal notation for an integer. The base terms *true* and *false* designate truth values. The base terms $\Diamond$ and $\mathcal{E}$ will be explained later, for now it suffices to say that they correspond to an empty list and an erroneous computation, respectively. The base terms designating truth values or integers are said to be *atomic base terms* since they are used to represent the *atoms* in the domain D. We could easily extend L to include more atomic base terms, we use the current set because it is the smallest set sufficient for the example correctness proofs in Sections 8 and 9.

The second and third lines of disjunctions give the *function symbols* of L. The infix function symbol ":" is referred to as *cons*. For notational convenience, *cons* associates to the right. Terms formed using only base terms and function symbols are called *simple terms*. Terms can also be formed using one of a countably infinite number of *function variables*, such as in the fourth line of disjunctions. If E is a term of the form $g(E_1, \ldots, E_r)$, where g is a function symbol or a function variable, then g is the *root* of E. Whenever we refer to a *variable* without specifying whether it is an object variable or a function variable, it is always an object variable. Thus, a *variable-free* term is a term that contains no object variables, but may contain function variables. A variable-free simple term contains neither object variables nor function variables.

The notation $E[v_1, \ldots, v_k]$ is used to designate a term that has its variables among $v_1, \ldots, v_k$. The term $E[E_1, \ldots, E_k]$ is formed from $E[v_1, \ldots, v_k]$ by replacing each instance of v_i by E_i for i between 1 and k. Let E' and E'' be terms. If there exists a term $E[v]$ that contains at least one instance of v such that E' is identical to $E[E'']$, then E'' is a *subterm* of E'. Notice that any term is a subterm of itself by this definition.

A program P in L consists of a sequence of declarations, followed by a variable-free term called the *value term* of P, written E_P. Declarations are of the form

$$f(v_1, \ldots, v_r) \Leftarrow E[v_1, \ldots, v_r],$$

where f is a function variable, $r \geq 0$ and the v_i are variables. Any function variable mentioned in a program must be declared exactly once. Any instances of a function

variable f, including that on the left side of the declaration of f, must have the same arity. Restricting programs to this simple form avoids issues of scoping and "first-class" functions. Notice that function variables may be declared to have an arity of zero, we will not write parentheses following an instance of such a function variable.

3. Strict and Non-Strict Semantics

In this section we describe strict and non-strict semantics for the functional programming language L. The semantics are described by giving a set of axioms for each: $\mathcal{S}$ for strict and $\mathcal{L}$ for non-strict. Some of these axioms will be in the form of definitions, but whether or not an axiom is in the form of a definition does not concern us, so in either case it is simply called an axiom. The axioms are numbered in sequence, and unless otherwise noted, each axiom belongs to both $\mathcal{S}$ and $\mathcal{L}$.

The semantic function $\mathcal{P}$ maps programs to their denotations in the domain D. The axioms also refer to an auxiliary semantic function $\mathcal{D}$, which takes two arguments, a variable-free term and an *environment*, and returns an element of D. An *environment* η is a finite partial function from function variables to functions over D, or, more precisely, an environment is an element of

$$F \xrightarrow{\text{fin}} \bigcup_{n=0}^{\infty} (D^n \to D),$$

where F is the set of function variables and $(D^n \to D)$ is the set of functions over D of arity n. The application of $\mathcal{D}$ to a variable-free term E and an environment η is written $\mathcal{D}[\![E]\!]\eta$. We use emphatic brackets whenever a function is applied to a term or a program in L.

Environments are used to represent the effects of declarations in a program. If a set of declarations results in an environment η, then $\mathcal{D}[\![E]\!]\eta$ is the denotation of the variable-free term E in the presence of these declarations. Notice that we have not placed any restriction on what kind of total function over D each $\eta(f)$ can be. In particular, we have not required that that each $\eta(f)$ be a continuous function. We could have assumed without loss of generality that $\eta(f)$ is always a continuous function, but it is instructive to actually prove that there would be no loss of generality (see the discussion following Axiom 15).

The value of $\mathcal{P}[\![P]\!]$ is $\mathcal{D}[\![E_P]\!]\eta$ where η is an environment derived from the declarations of P. Formalizing this requires specifying $\mathcal{D}$, and also specifying how environments depend on declarations. We begin by stating properties required of the semantic function $\mathcal{D}$. The first is a property of D, the codomain of $\mathcal{D}$. Let the set of *atoms* A be the union of the set of non-negative integers and the set of boolean values $\{true, false\}$. Let $\mathcal{A}$ be the bijection from atomic base terms to the corresponding elements of A.

Axiom 1. (Semantic Classes) The objects of D are divided into two disjoint sets, the *unconstructed objects* and the *constructed objects*. The unconstructed objects are $\perp^D$, $\mathcal{E}^D$, $\Diamond^D$ and the members of the set of atoms A.

To specify $\mathcal{D}$, it is helpful to extend it to be defined over *generalized terms*. The set of *generalized base terms* is the set of base terms together with the elements of D. The formation of *generalized terms* from generalized base terms is the same as the formation of (non-generalized) terms. We assume that the set of terms and the domain $\mathcal{D}$ are such that any generalized term can be formed in exactly one way. If E is some (non-generalized) term, we sometimes emphasize this by refering to E as a *program term*.

We specify $\mathcal{D}$ by induction on the structure of variable-free generalized terms. We begin with variable-free generalized base terms.

Axiom 2. (Base Terms) If E is a variable-free generalized base term, then for any environment η

$$\mathcal{D}[\![E]\!]\eta = \begin{cases} \mathcal{E}^D & \text{if } E = \mathcal{E}; \\ \Diamond^D & \text{if } E = \Diamond; \\ \mathcal{A}[\![E]\!] & \text{if } E \text{ is an atomic base term;} \\ E & \text{if } E \text{ is an object in } D. \end{cases}$$

We often simply write $\perp$, $\mathcal{E}$ and $\Diamond$ for $\perp^D$, $\mathcal{E}^D$ and $\Diamond^D$. Next we must specify $\mathcal{D}$ for non-base variable-free generalized terms. This requires the following two axioms.

Axiom 3. (Function Symbols) Let g be a function symbol of arity r. There exists a function $g^{\mathcal{D}}$ (the *interpretation of g with respect to $\mathcal{D}$*) from D^r to D such that

$$\mathcal{D}[\![g(E_1, \ldots, E_k)]\!]\eta = g^{\mathcal{D}}(\mathcal{D}[\![E_1]\!]\eta, \ldots, \mathcal{D}[\![E_k]\!]\eta)$$

if $k = r$ and each of the $\mathcal{D}[\![E_i]\!]\eta$ is defined. Otherwise, $\mathcal{D}[\![g(E_1, \ldots, E_k)]\!]\eta$ is undefined.

Axiom 4. (Function Variables) Let f be a function variable. Then

$$\mathcal{D}[\![f(E_1, \ldots, E_k)]\!]\eta = \eta(f)(\mathcal{D}[\![E_1]\!]\eta, \ldots, \mathcal{D}[\![E_k]\!]\eta)$$

if each of the $\mathcal{D}[\![E_i]\!]\eta$ is defined, η is defined on f, and $\eta(f)$ is a function of arity k. Otherwise, $\mathcal{D}[\![f(E_1, \ldots, E_k)]\!]\eta$ is undefined.

If η is an environment, then a generalized term E is *meaningful in η* if and only if $\mathcal{D}[\![E]\!]\eta$ is defined. Let S be a set of environments. We define a binary relation $\equiv_S$ on generalized terms such that $E_1 \equiv_S E_2$ (E_1 and E_2 are *equivalent relative to S*) if and only if $\mathcal{D}[\![E_1]\!]\eta = \mathcal{D}[\![E_2]\!]\eta$ for all environments η in S. We usually abbreviate the assertion $E_1 \equiv_S E_2$ by writing $E_1 \equiv E_2$ (E_1 and E_2 are *equivalent*). With this notation, unless otherwise noted, it is understood that the set S is the set of all environments. Notice that if x and y are elements of D (and are, therefore, generalized terms), then $x = y$ if and only if $x \equiv y$. We will often write $x \equiv y$ instead of $x = y$ to emphasize that objects x and y in D can be viewed as generalized terms.

To complete the specification of $\mathcal{D}$, we must specify $g^{\mathcal{D}}$ for each function symbol g. We first specify the interpretation of the function symbol *cons*. The following axiom is the only axiom that differs between $\mathcal{S}$ and $\mathcal{L}$.

Axiom 5 of $\mathcal{S}$. (*cons*) For all x and y in D,

$$x : y \equiv \begin{cases} \perp & \text{if } x \equiv \perp; \\ \mathcal{E} & \text{if } x \equiv \mathcal{E}; \\ \perp & \text{if } x \not\equiv \mathcal{E} \text{ and } y \equiv \perp; \\ \mathcal{E} & \text{if } x \not\equiv \perp \text{ and } y \equiv \mathcal{E}; \\ \mathcal{E} & \text{if } x \not\equiv \perp \text{ and } y \text{ is an atom;} \\ \text{a constructed object} & \text{otherwise.} \end{cases}$$

Axiom 5 of $\mathcal{L}$. (*cons*) For all x and y in D, the generalized term $x : y$ is equivalent to a constructed object.

In both $\mathcal{S}$ and $\mathcal{L}$, Axiom 5 does not completely determine the interpretation of *cons*, since specifying that $x : y$ is a constructed object does not specify which constructed object it is. This is remedied by the Constructability Axiom.

Axiom 6. (Constructability) For all constructed objects z, there exist unique x and y such that $z \equiv x : y$.

The interpretation of *cons* required by Axioms 5 and 6 of $\mathcal{S}$ is consistent with the behavior of standard list processing languages that use eager-evaluation. The choice of Axiom 5 of $\mathcal{L}$ was influenced by implementation issues that are discussed near the end of Section 7.

The semantics of *cons* given by the above axioms can be described using more conventional methods than those used here. However, axiomatizing *cons* in the above way allows the fundemental differences between the strict and the non-strict semantics to be isolated in one axiom. This makes the similiarities between the two semantics more clear.

Clearly *cons* is a non-strict function in both $\mathcal{S}$ and $\mathcal{L}$. Why is only the semantics given by $\mathcal{L}$ called non-strict? A function h of arity r is *non-strict* if there exists x_i for $1 \leq i \leq r$ such that for some k

$$h(x_1, \ldots, x_{k-1}, \perp, x_{k+1}, x_r) \neq \perp,$$

otherwise h is *strict*. The function h is *trivially non-strict* if for any such $\{x_i\}$ and k

$$h(x_1, \ldots, x_{k-1}, x_k, x_{k+1}, x_r) = h(x_1, \ldots, x_{k-1}, \perp, x_{k+1}, x_r),$$

otherwise h is *non-trivially non-strict*. We call a semantics non-strict if and only there is a function symbol whose interpretation in that semantics is a non-trivially non-strict function. Thus $\mathcal{L}$ is a non-strict semantics because the interpretation of *cons* in $\mathcal{L}$ is non-trivially non-strict. We shall see that in $\mathcal{S}$ all function symbols have either strict or trivially non-strict functions as interpretations, so $\mathcal{S}$ is a strict semantics.

The following six axioms specify the interpretations of the remaining function symbols in L.

Axiom 7. (*hd* **and** *tl*) For all z in D, if $x:y$ is equivalent to a constructed object, then

$$hd(z) \equiv \begin{cases} \bot & \text{if } z \equiv \bot; \\ \mathcal{E} & \text{if } z \equiv \diamondsuit \text{ or } z \equiv \mathcal{E}; \\ \mathcal{E} & \text{if } z \text{ is an atom;} \\ x & \text{if } z \equiv x:y; \end{cases} \qquad tl(z) \equiv \begin{cases} \bot & \text{if } z \equiv \bot; \\ \mathcal{E} & \text{if } z \equiv \diamondsuit \text{ or } z \equiv \mathcal{E}; \\ \mathcal{E} & \text{if } z \text{ is an atom;} \\ y & \text{if } z \equiv x:y. \end{cases}$$

Axiom 8. (Conditional) For all x, y, and z in D,

$$\textbf{if } x \textbf{ then } y \textbf{ else } z \equiv \begin{cases} \bot & \text{if } x \equiv \bot; \\ y & \text{if } x \equiv true; \\ z & \text{if } x \equiv false; \\ \mathcal{E} & \text{otherwise.} \end{cases}$$

Axiom 9. (*atom* **and** *nil* **predicates)** For all x in D,

$$atom(x) \equiv \begin{cases} \bot & \text{if } x \equiv \bot; \\ \mathcal{E} & \text{if } x \equiv \mathcal{E}; \\ true & \text{if } x \text{ is an atom;} \\ false & \text{otherwise;} \end{cases} \qquad nil(x) \equiv \begin{cases} \bot & \text{if } x \equiv \bot; \\ \mathcal{E} & \text{if } x \equiv \mathcal{E}; \\ true & \text{if } x = \diamondsuit; \\ false & \text{otherwise.} \end{cases}$$

Axiom 10. (+, −, **and** ×) For all x and y in D, if $*$ designates either +, −, or ×, then

$$x * y \equiv \begin{cases} \bot & \text{if } x \equiv \bot; \\ \bot & \text{if } x \text{ is an integer and } y \equiv \bot; \\ x * y & \text{if } x \text{ and } y \text{ are integers;} \\ \mathcal{E} & \text{otherwise.} \end{cases}$$

Axiom 11. (mod) For all x and y in D,

$$x \bmod y \equiv \begin{cases} \bot & \text{if } x \equiv \bot; \\ \bot & \text{if } x \text{ is an integer and } y \equiv \bot; \\ x \bmod y & \text{if } x \text{ and } y \text{ are integers and } y > 0; \\ \mathcal{E} & \text{otherwise.} \end{cases}$$

Axiom 12. (= **and** ≤) For all x and y in D, if $*$ designates either = or ≤, then

$$x * y \equiv \begin{cases} \bot & \text{if } x \equiv \bot; \\ \bot & \text{if } x \text{ is an integer and } y \equiv \bot; \\ x * y & \text{if } x \text{ and } y \text{ are integers;} \\ \mathcal{E} & \text{otherwise.} \end{cases}$$

In Axioms 10 and 12 the symbols +, −, ×, =, and ≤ have different meanings depending on whether they are on the left or right side of the ≡. On the left side they are symbols in L, on the right side they designate the standard function or predicate over the integers.

We have now completed the specification of $\mathcal{D}$ (relative to D), and can prove the following basic theorems.

Theorem 3-1. Let E be a variable-free generalized term and η an environment. Then E is meaningful in η if and only if η is defined on every function variable f in E, and $\eta(f)$ has the same arity as each instance of f in E.

Proof: The proof is by induction on the structure of E. Recall that E is meaningful in η if and only if $\mathcal{D}[\![E]\!]\eta$ is defined. By Axiom 2, the theorem clearly holds when E is a generalized base term. The induction step follows from Axioms 3 and 4. □

Theorem 3-2. Let F be a set of function variables, and let η and η' be environments such that $\eta(f) = \eta'(f)$ for every function variable f not in F. Let E be a variable-free generalized term meaningful in η that does not mention any of the function variables in F. Then E is meaningful in η' and $\mathcal{D}[\![E]\!]\eta = \mathcal{D}[\![E]\!]\eta'$.

Proof: The proof is by induction on the structure of E. The theorem clearly holds when E is a base term. If E has a function symbol g as its root, then

$$\begin{aligned}
\mathcal{D}[\![g(E_1,\ldots,E_r)]\!]\eta &= g^{\mathcal{D}}(\mathcal{D}[\![E_1]\!]\eta,\ldots,\mathcal{D}[\![E_r]\!]\eta) && \text{Axiom 3}\\
&= g^{\mathcal{D}}(\mathcal{D}[\![E_1]\!]\eta',\ldots,\mathcal{D}[\![E_r]\!]\eta') && \text{ind. hyp.}\\
&= \mathcal{D}[\![g(E_1,\ldots,E_r)]\!]\eta' && \text{Axiom 3.}
\end{aligned}$$

If E has a function variable f not in F as its root, then

$$\begin{aligned}
\mathcal{D}[\![f(E_1,\ldots,E_r)]\!]\eta &= \eta(f)(\mathcal{D}[\![E_1]\!]\eta,\ldots,\mathcal{D}[\![E_r]\!]\eta) && \text{Axiom 4}\\
&= \eta(f)(\mathcal{D}[\![E_1]\!]\eta',\ldots,\mathcal{D}[\![E_r]\!]\eta') && \text{ind. hyp.}\\
&= \eta'(f)(\mathcal{D}[\![E_1]\!]\eta',\ldots,\mathcal{D}[\![E_r]\!]\eta') && \text{given}\\
&= \mathcal{D}[\![f(E_1,\ldots,E_r)]\!]\eta' && \text{Axiom 4.}
\end{aligned}$$ □

Theorem 3-3 states a property that is a consequence of the extending of $\mathcal{D}$ to generalized terms. This property will be used in many of the proofs that follow, and is the primary reason for introducing generalized terms.

Theorem 3-3. Let η be an environment. Let $E[v_1,\ldots,v_k]$ be a generalized term, and let $E'_1,\ldots,E'_k$ be variable-free generalized terms such that $E[E'_1,\ldots,E'_k]$ is meaningful in η. Then

$$\mathcal{D}[\![E[E'_1,\ldots,E'_k]]\!]\eta = \mathcal{D}[\![E[\mathcal{D}[\![E'_1]\!]\eta,\ldots,\mathcal{D}[\![E'_k]\!]\eta]]\!]\eta.$$

Proof: The proof is by induction on the structure of E. The base case follows since $\mathcal{D}[\![x]\!]\eta = x$ for any x in D. To prove the induction step, we assume, without loss of generality, that $E[v_1,\ldots,v_n]$ is of the form

$$g(E_1[v_1,\ldots,v_n],\ldots,E_r[v_1,\ldots,v_n]),$$

where g is a function symbol or a function variable. If the root of E is a function

symbol g, then

$$\begin{aligned}
&\mathcal{D}[\![E[E'_1,\ldots,E'_k]]\!]\eta \\
&\quad = \mathcal{D}[\![g(E_1[E'_1,\ldots,E'_k],\ldots,E_r[E'_1,\ldots,E'_k])]\!]\eta && \text{given} \\
&\quad = g^{\mathcal{D}}(\mathcal{D}[\![E_1[E'_1,\ldots,E'_k]]\!]\eta,\ldots,\mathcal{D}[\![E_r[E'_1,\ldots,E'_k]]\!]\eta) && \text{Axiom 3} \\
&\quad = g^{\mathcal{D}}(\mathcal{D}[\![E_1[\mathcal{D}[\![E'_1]\!]\eta,\ldots,\mathcal{D}[\![E'_k]\!]\eta]]\!]\eta,\ldots, && \text{ind. hyp.} \\
&\qquad \mathcal{D}[\![E_r[\mathcal{D}[\![E'_1]\!]\eta,\ldots,\mathcal{D}[\![E'_k]\!]\eta]]\!]\eta) \\
&\quad = \mathcal{D}[\![g(E_1[\mathcal{D}[\![E'_1]\!]\eta,\ldots,\mathcal{D}[\![E'_k]\!]\eta],\ldots, && \text{Axiom 3} \\
&\qquad E_r[\mathcal{D}[\![E'_1]\!]\eta,\ldots,\mathcal{D}[\![E'_k]\!]\eta])]\!]\eta \\
&\quad = \mathcal{D}[\![E[\mathcal{D}[\![E'_1]\!]\eta,\ldots,\mathcal{D}[\![E'_k]\!]\eta]]\!]\eta && \text{given.}
\end{aligned}$$

If the root of E is a function variable f, then

$$\begin{aligned}
&\mathcal{D}[\![E[E'_1,\ldots,E'_k]]\!]\eta \\
&\quad = \mathcal{D}[\![f(E_1[E'_1,\ldots,E'_k],\ldots,E_r[E'_1,\ldots,E'_k])]\!]\eta && \text{given} \\
&\quad = \eta(f)(\mathcal{D}[\![E_1[E'_1,\ldots,E'_k]]\!]\eta,\ldots,\mathcal{D}[\![E_r[E'_1,\ldots,E'_k]]\!]\eta) && \text{Axiom 4} \\
&\quad = \eta(f)(\mathcal{D}[\![E_1[\mathcal{D}[\![E'_1]\!]\eta,\ldots,\mathcal{D}[\![E'_k]\!]\eta]]\!]\eta,\ldots, && \text{ind. hyp.} \\
&\qquad \mathcal{D}[\![E_r[\mathcal{D}[\![E'_1]\!]\eta,\ldots,\mathcal{D}[\![E'_k]\!]\eta]]\!]\eta) \\
&\quad = \mathcal{D}[\![f(E_1[\mathcal{D}[\![E'_1]\!]\eta,\ldots,\mathcal{D}[\![E'_k]\!]\eta],\ldots, && \text{Axiom 4} \\
&\qquad E_r[\mathcal{D}[\![E'_1]\!]\eta,\ldots,\mathcal{D}[\![E'_k]\!]\eta])]\!]\eta \\
&\quad = \mathcal{D}[\![E[\mathcal{D}[\![E'_1]\!]\eta,\ldots,\mathcal{D}[\![E'_k]\!]\eta]]\!]\eta && \text{given.} \quad \square
\end{aligned}$$

To complete the specification of our semantics, we must specify how environments depend on declarations. The first step in this is to require a partial order $\sqsubseteq$ over D, the *information ordering* over D.

Axiom 13. ($\sqsubseteq$)

a) For all x, $x \sqsubseteq x$. (reflexivity)

b) For all x and y, if $x \sqsubseteq y$ and $y \sqsubseteq x$, then $x \equiv y$. (antisymmetry)

c) For all x, y, and z, if $x \sqsubseteq y$ and $y \sqsubseteq z$, then $x \sqsubseteq z$. (transitivity)

d) If x or y is an unconstructed object, then $x \sqsubseteq y$ if and only if $x \equiv \perp$ or $x \equiv y$.

e) If x and y are constructed objects, then $x \sqsubseteq y$ if and only if $hd(x) \sqsubseteq hd(y)$ and $tl(x) \sqsubseteq tl(y)$.

Parts (a) through (c) of the above axiom are the properties that $\sqsubseteq$ must have to be a partial order. These properties must be axiomatized explicitly because they do not follow from (d) and (e).

We also use the symbol $\sqsubseteq$ for a partial order over the set of functions over D, and also for a partial order over environments. If f and g are functions of different arity, then $f \not\sqsubseteq g$. If functions f and g have arity r, then $f \sqsubseteq g$ if and only if

$$f(x_1, x_2, \ldots, x_r) \sqsubseteq g(x_1, x_2, \ldots, x_r)$$

for all $x_1, x_2, \ldots, x_r$ in D. If η_1 and η_2 are environments with identical domains, then $\eta_1 \sqsubseteq \eta_2$ if $\eta_1(f) \sqsubseteq \eta_2(f)$ for all function variables f for which the two environments are defined. Otherwise, $\eta_1 \not\sqsubseteq \eta_2$.

Let S be a set of environments. We define a binary relation $\sqsubseteq_S$ on generalized terms such that $E_1 \sqsubseteq_S E_2$ if and only if $\mathcal{D}[\![E_1]\!]\eta \sqsubseteq \mathcal{D}[\![E_2]\!]\eta$ for all environments η in S. Thus, $\sqsubseteq_S$ is a partial ordering relative to $\equiv_S$ for any set of environments S. We usually abbreviate the assertion $E_1 \sqsubseteq_S E_2$ by writing $E_1 \sqsubseteq E_2$. With this notation, unless otherwise noted, it is understood that the set S is the set of all environments.

The existence of the information ordering allows us to state the following requirement on the domain D.

Axiom 14. (Closure) For all chains $\{x_j\}$ of objects in D, there exists an object x such that

$$x = \bigsqcup_{j=0}^{\infty} x_j.$$

For the remainder of this section we shall assume, without loss of generality, that any program P has n declarations of the form

$$f_k(v_1, \ldots, v_{r_k}) \Leftarrow R_k[v_1, \ldots, v_{r_k}]$$

for $1 \leq k \leq n$. We write F_P for the set of function variables $\{f_k \mid 1 \leq k \leq n\}$. We also write E_P for the value term of P.

An environment η is called a *fixed point of the declarations of* P if

$$\mathcal{D}[\![f_k(x_1, \ldots, x_{r_k})]\!]\eta = \mathcal{D}[\![R_k[x_1, \ldots, x_{r_k}]]\!]\eta$$

for all k between 1 and n and all objects $x_1, \ldots, x_{r_k}$, and if $\eta(f)$ is undefined for all f not in F_P. We would like the environment produced by the declarations of P to be a fixed point of those declarations. This is the intuitive meaning we have in mind when we write declarations for a program. There may be many environments with this property, we choose the least such. If η is a fixed point of the declarations of P and $\eta \sqsubseteq \eta'$ for any η' that is also a fixed point of the declarations of P, then η is the *least fixed point of the declarations of* P.

We prove in Theorem 3-12 that for any program P, the least fixed point of the declarations of P exists. The following theorems and definitions are preliminary results necessary for the proof of Theorem 3-12. The reader may skip to after Theorem 3-12 if not interested in its proof.

The first definition needed is of the function N_P that has as its domain and codomain the set of environments defined only on F_P, and is such that

$$N_P(\eta)(f_k) = \lambda x_1, \ldots, x_{r_k}.\mathcal{D}[\![R_k[x_1, \ldots, x_{r_k}]]\!]\eta$$

for all k between 1 and n, where the x_i range over D. The usefulness of this function in shown in the next theorem.

Theorem 3-4. For any program P, an environment η is a fixed point of N_P if and only if it is a fixed point of the declarations of P.

Proof: Clearly fixed points of the declarations of P and the fixed points of N_P both have F_P as their domain. To prove the forward implication, assume that η is a fixed point of N_P. Then, for all k between 1 and n,

$$\begin{aligned}
\mathcal{D}[\![f_k(E_1, \ldots, E_{r_k})]\!]\eta &= \eta(f_k)(\mathcal{D}[\![E_1]\!]\eta, \ldots, \mathcal{D}[\![E_{r_k}]\!]\eta) && \text{Axiom 4} \\
&= N_P(\eta)(f_k)(\mathcal{D}[\![E_1]\!]\eta, \ldots, \mathcal{D}[\![E_{r_k}]\!]\eta) && \text{given} \\
&= \mathcal{D}[\![R_k[\mathcal{D}[\![E_1]\!]\eta, \ldots, \mathcal{D}[\![E_{r_k}]\!]\eta]]\!]\eta && \text{def. of } N_P \\
&= \mathcal{D}[\![R_k[E_1, \ldots, E_{r_k}]]\!]\eta && \text{Theorem 3-3.}
\end{aligned}$$

Therefore, η is also a fixed point of the declarations of P. To prove the reverse implication, assume that η is a fixed point of the declarations of P. Then for all k between 1 and n,

$$\begin{aligned}
N_P(\eta)(f_k)(x_1, \ldots, x_{r_k}) &= \mathcal{D}[\![R_k[x_1, \ldots, x_{r_k}]]\!]\eta && \text{def. of } N_P \\
&= \mathcal{D}[\![f_k(x_1, \ldots, x_{r_k})]\!]\eta && \text{given} \\
&= \eta(f_k)(x_1, \ldots, x_{r_k}) && \text{Axiom 4.}
\end{aligned}$$

Therefore, η is also a fixed point of N_P. □

An environment η is an *environment of monotonic functions* if for all f for which $\eta(f)$ is defined, $\eta(f)$ is a monotonic function. An *environment of continuous functions* is similarly defined.

Theorem 3-5. Let η be an environment and $E[v_1, \ldots, v_n]$ be a generalized term such that $E[x_1, \ldots, x_n]$ is meaningful in η for all x_i in D. Consider the function

$$p = \lambda x_1, \ldots, x_n.\mathcal{D}[\![E[x_1, \ldots, x_n]]\!]\eta.$$

If η is an environment of monotonic functions, then p is monotonic. If η is an environment of continuous functions, then p is continuous.

Proof: The proof is by induction on the structure of E. The theorem clearly holds when E is a base term, since in that case p is either a constant function or a projection function.

To prove the induction step, we will assume that $g^{\mathcal{D}}$ is a continuous function for every function symbol g. We will come back to this assumption later in the proof. We also assume, without loss of generality, that $E[v_1, \ldots, v_n]$ is of the form

$$g(E_1[v_1,\ldots,v_n],\ldots,E_r[v_1,\ldots,v_n]),$$

where g is a function symbol or a function variable. To prove the first implication, let η be an environment of monotonic functions, and let $x_1,\ldots,x_n$ and $y_1,\ldots,y_n$ be elements of D such that $x_k \sqsubseteq y_k$ for each k. If the root of $E[v_1,\ldots,v_n]$ is a function symbol g, then

$$\begin{aligned}
&p(x_1,\ldots,x_n)\\
&\quad= \mathcal{D}[\![g(E_1[x_1,\ldots,x_n],\ldots,E_r[x_1,\ldots,x_n])]\!]\eta && \text{given}\\
&\quad= g^{\mathcal{D}}(\mathcal{D}[\![E_1[x_1,\ldots,x_n]]\!]\eta,\ldots,\mathcal{D}[\![E_r[x_1,\ldots,x_n]]\!]\eta) && \text{Axiom 3}\\
&\quad\sqsubseteq g^{\mathcal{D}}(\mathcal{D}[\![E_1[y_1,\ldots,y_n]]\!]\eta,\ldots,\mathcal{D}[\![E_r[y_1,\ldots,y_n]]\!]\eta) && \text{ind. hyp. and}\\
&\qquad && \text{mon. of } g^{\mathcal{D}}\\
&\quad= \mathcal{D}[\![g(E_1[y_1,\ldots,y_n],\ldots,E_r[y_1,\ldots,y_n])]\!]\eta && \text{Axiom 3}\\
&\quad= p(y_1,\ldots,y_n) && \text{given.}
\end{aligned}$$

If the root of $E[v_1,\ldots,v_n]$ is a function variable f, then

$$\begin{aligned}
&p(x_1,\ldots,x_n)\\
&\quad= \mathcal{D}[\![f(E_1[x_1,\ldots,x_n],\ldots,E_r[x_1,\ldots,x_n])]\!]\eta && \text{given}\\
&\quad= \eta(f)(\mathcal{D}[\![E_1[x_1,\ldots,x_n]]\!]\eta,\ldots,\mathcal{D}[\![E_r[x_1,\ldots,x_n]]\!]\eta) && \text{Axiom 4}\\
&\quad\sqsubseteq \eta(f)(\mathcal{D}[\![E_1[y_1,\ldots,y_n]]\!]\eta,\ldots,\mathcal{D}[\![E_r[y_1,\ldots,y_n]]\!]\eta) && \text{ind. hyp. and}\\
&\qquad && \text{mon. of } \eta(f)\\
&\quad= \mathcal{D}[\![f(E_1[y_1,\ldots,y_n],\ldots,E_r[y_1,\ldots,y_n])]\!]\eta && \text{Axiom 3}\\
&\quad= p(y_1,\ldots,y_n) && \text{given.}
\end{aligned}$$

To prove the second implication, let η be an environment of continuous functions, and for each k between 1 and n let $\{x_{k,j}\}$ be a chain with x_k as its least upper bound. If the root of $E[v_1,\ldots,v_n]$ is a function symbol g, then

$$\begin{aligned}
&p(x_1,\ldots,x_n)\\
&\quad= \mathcal{D}[\![g(E_1[x_1,\ldots,x_n],\ldots,E_r[x_1,\ldots,x_n])]\!]\eta && \text{given}\\
&\quad= g^{\mathcal{D}}(\mathcal{D}[\![E_1[x_1,\ldots,x_n]]\!]\eta,\ldots,\mathcal{D}[\![E_r[x_1,\ldots,x_n]]\!]\eta) && \text{Axiom 3}\\
&\quad= g^{\mathcal{D}}(\textstyle\bigsqcup_{j=0}^{\infty}\mathcal{D}[\![E_1[x_{1,j},\ldots,x_{n,j}]]\!]\eta,\ldots, && \text{ind. hyp.}\\
&\qquad \textstyle\bigsqcup_{j=0}^{\infty}\mathcal{D}[\![E_r[x_{1,j},\ldots,x_{n,j}]]\!]\eta)\\
&\quad= \textstyle\bigsqcup_{j=0}^{\infty} g^{\mathcal{D}}(\mathcal{D}[\![E_1[x_{1,j},\ldots,x_{n,j}]]\!]\eta,\ldots, && \text{cont. of } g^{\mathcal{D}}\\
&\qquad \mathcal{D}[\![E_r[x_{1,j},\ldots,x_{n,j}]]\!]\eta)\\
&\quad= \textstyle\bigsqcup_{j=0}^{\infty}\mathcal{D}[\![g(E_1[x_{1,j},\ldots,x_{n,j}],\ldots,E_r[x_{1,j},\ldots,x_{n,j}])]\!]\eta && \text{Axiom 3}\\
&\quad= \textstyle\bigsqcup_{j=0}^{\infty} p(x_{1,j},\ldots,x_{n,j}) && \text{given.}
\end{aligned}$$

If the root of $E[v_1, \ldots, v_n]$ is a function variable f, then

$$
\begin{aligned}
& p(x_1, \ldots, x_n) \\
& \quad = \mathcal{D}[\![f(E_1[x_1, \ldots, x_n], \ldots, E_r[x_1, \ldots, x_n])]\!]\eta && \text{given} \\
& \quad = \eta(f)(\mathcal{D}[\![E_1[x_1, \ldots, x_n]]\!]\eta, \ldots, \mathcal{D}[\![E_r[x_1, \ldots, x_n]]\!]\eta) && \text{Axiom 4} \\
& \quad = \eta(f)(\textstyle\bigsqcup_{j=0}^{\infty} \mathcal{D}[\![E_1[x_{1,j}, \ldots, x_{n,j}]]\!]\eta, \ldots, && \text{ind. hyp.} \\
& \qquad \textstyle\bigsqcup_{j=0}^{\infty} \mathcal{D}[\![E_r[x_{1,j}, \ldots, x_{n,j}]]\!]\eta) \\
& \quad = \textstyle\bigsqcup_{j=0}^{\infty} \eta(f)(\mathcal{D}[\![E_1[x_{1,j}, \ldots, x_{n,j}]]\!]\eta, \ldots, && \text{cont. of } \eta(f) \\
& \qquad \mathcal{D}[\![E_r[x_{1,j}, \ldots, x_{n,j}]]\!]\eta) \\
& \quad = \textstyle\bigsqcup_{j=0}^{\infty} \mathcal{D}[\![f(E_1[x_{1,j}, \ldots, x_{n,j}], \ldots, E_r[x_{1,j}, \ldots, x_{n,j}])]\!]\eta && \text{Axiom 4} \\
& \quad = \textstyle\bigsqcup_{j-0}^{\infty} p(x_{1,j}, \ldots, x_{n,j}) && \text{given.}
\end{aligned}
$$

To complete the proof, we must justify our assumption that each $g^{\mathcal{D}}$ is a continuous function. We will prove this for the function symbol *cons*, the remaining function symbols are left as an exercise for the reader.

For the proof that *cons* is monotonic in $\mathcal{S}$ let $x \sqsubseteq x'$ and $y \sqsubseteq y'$ be elements of D. We must show that $x : y \sqsubseteq x' : y'$. This clearly holds when $x : y \equiv \perp$. If $x : y \equiv \mathcal{E}$ (either because $x \equiv \mathcal{E}$, or because $x \not\equiv \perp$ and y is an atom or $\mathcal{E}$), then $x' : y'$ is also equivalent to $\mathcal{E}$. In the final case $x : y$ is equivalent to a constructed object, and, therefore, so is $x' : y'$. Thus, $x : y \sqsubseteq x' : y'$ by Axiom 13.

The proof that *cons* is monotonic in $\mathcal{L}$ is much easier. Since $x : y$ and $x' : y'$ are always equivalent to constructed objects, it follows from Axiom 13 that $x : y \sqsubseteq x' : y'$.

For the proof that *cons* is continuous in $\mathcal{S}$ let $\{x_k\}$ and $\{y_k\}$ be chains with least upper bounds of x and y, respectively. Also, let $z_k \equiv x_k : y_k$ for all k. Notice that by the monotonicity of *cons*, $\{z_k\}$ is a chain as well. Let z be its least upper bound. We must show that $z \equiv x : y$. If $x : y \equiv \perp$ (either because $x \equiv \perp$ or because $y \equiv \perp$ and $x \not\equiv \mathcal{E}$), then $z_k \equiv \perp$ for each k, so $z \equiv \perp$. If $x : y \equiv \mathcal{E}$ (either because $x \equiv \mathcal{E}$ or because $x \not\equiv \perp$ and y is an atom or $\mathcal{E}$), then there exists a k such that $z_k \equiv \mathcal{E}$, so $z \equiv \mathcal{E}$. In the final case $x : y$ is a constructed object. In this case, there exists a k such that z_k is a constructed object, therefore, z is a constructed object. We show that $z \equiv x : y$ by showing that

$$x : y \sqsubseteq z' \iff \forall k[z_k \sqsubseteq z']$$

for all z'. This condition clearly holds when z' is an unconstructed object since both sides of the if and only if are false. If z' is a constructed object then

$$
\begin{aligned}
x : y \sqsubseteq z' & \iff (\textstyle\bigsqcup_{k=0}^{\infty} x_k) : (\bigsqcup_{k=0}^{\infty} y_k) \sqsubseteq z' && \text{def. of } x \text{ and } y \\
& \iff \textstyle\bigsqcup_{k=0}^{\infty} x_k \sqsubseteq hd(z') \wedge \bigsqcup_{k=0}^{\infty} y_k \sqsubseteq tl(z') && \text{Axiom 13} \\
& \iff \forall k[x_k \sqsubseteq hd(z')] \wedge \forall k[y_k \sqsubseteq tl(z')] && \text{def. of } \textstyle\bigsqcup \\
& \iff \forall k[x_k \sqsubseteq hd(z') \wedge y_k \sqsubseteq tl(z')] && \text{prop. of } \forall \\
& \iff \forall k[z_k \sqsubseteq z'] && \text{Axiom 13.}
\end{aligned}
$$

The proof that *cons* is continuous in $\mathcal{L}$ is very similar to the proof that *cons* is continuous in $\mathcal{S}$ when *cons* returns a constructed object. We leave it as an exercise for the reader. □

Theorem 3-6. Let E be a variable-free generalized term. The restriction of

$$\lambda\eta.\mathcal{D}[\![E]\!]\eta$$

to environments of continuous functions in which E is meaningful is a continuous function.

Proof: We first prove that the function is monotonic by induction on the structure of E. The base case holds trivially since constant functions are monotonic. To prove the induction step, let $\eta \sqsubseteq \eta'$ be environments of continuous functions in which E is meaningful. If E has a function symbol g as its root, then

$$\begin{aligned}
\mathcal{D}[\![g(E_1,\ldots,E_r)]\!]\eta &= \mathcal{D}[\![g(\mathcal{D}[\![E_1]\!]\eta,\ldots,\mathcal{D}[\![E_r]\!]\eta)]\!]\eta && \text{Theorem 3-3}\\
&\sqsubseteq \mathcal{D}[\![g(\mathcal{D}[\![E_1]\!]\eta',\ldots,\mathcal{D}[\![E_r]\!]\eta')]\!]\eta && \text{ind. hyp. and}\\
& && \text{Theorem 3-5}\\
&= \mathcal{D}[\![g(\mathcal{D}[\![E_1]\!]\eta',\ldots,\mathcal{D}[\![E_r]\!]\eta')]\!]\eta' && \text{Theorem 3-2}\\
&= \mathcal{D}[\![g(E_1,\ldots,E_r)]\!]\eta' && \text{Theorem 3-3.}
\end{aligned}$$

If E has a function variable f as its root, then

$$\begin{aligned}
\mathcal{D}[\![f(E_1,\ldots,E_r)]\!]\eta &= \eta(f)(\mathcal{D}[\![E_1]\!]\eta,\ldots,\mathcal{D}[\![E_r]\!]\eta) && \text{Axiom 4}\\
&\sqsubseteq \eta(f)(\mathcal{D}[\![E_1]\!]\eta',\ldots,\mathcal{D}[\![E_r]\!]\eta') && \text{ind. hyp. and}\\
& && \text{monotonicity of } \eta(f)\\
&\sqsubseteq \eta'(f)(\mathcal{D}[\![E_1]\!]\eta',\ldots,\mathcal{D}[\![E_r]\!]\eta') && \eta \sqsubseteq \eta'\\
&= \mathcal{D}[\![f(E_1,\ldots,E_r)]\!]\eta' && \text{Axiom 4.}
\end{aligned}$$

To prove continuity, let $\{\eta_j\}$ be a chain of environments of continuous functions in which E is meaningful. Let η be the least upper bound of $\{\eta_j\}$. We know by Theorem 3-1 and the definition of $\sqsubseteq$ on environments that η is an environment of continuous functions in which E is meaningful. The proof is by induction on the structure of E. The base case holds trivially since constant functions are continuous. If E has a function symbol g as its root, then

$$\begin{aligned}
&\textstyle\bigsqcup_{j=0}^{\infty} \mathcal{D}[\![g(E_1,\ldots,E_r)]\!]\eta_j\\
&\quad= \textstyle\bigsqcup_{j=0}^{\infty} \mathcal{D}[\![g(\mathcal{D}[\![E_1]\!]\eta_j,\ldots,\mathcal{D}[\![E_r]\!]\eta_j)]\!]\eta_j && \text{Theorem 3-3}\\
&\quad= \textstyle\bigsqcup_{j=0}^{\infty} \mathcal{D}[\![g(\mathcal{D}[\![E_1]\!]\eta_j,\ldots,\mathcal{D}[\![E_r]\!]\eta_j)]\!]\eta && \text{Theorem 3-2}\\
&\quad= \mathcal{D}[\![g(\textstyle\bigsqcup_{j=0}^{\infty} \mathcal{D}[\![E_1]\!]\eta_j,\ldots,\bigsqcup_{j=0}^{\infty} \mathcal{D}[\![E_r]\!]\eta_j)]\!]\eta && \text{Theorem 3-5}\\
&\quad= \mathcal{D}[\![g(\mathcal{D}[\![E_1]\!]\eta,\ldots,\mathcal{D}[\![E_r]\!]\eta)]\!]\eta && \text{ind. hyp.}\\
&\quad= \mathcal{D}[\![g(E_1,\ldots,E_r)]\!]\eta && \text{Theorem 3-3.}
\end{aligned}$$

If E has a function variable f as its root, then

$$\begin{aligned}
&\textstyle\bigsqcup_{j=0}^{\infty} \mathcal{D}[\![f(E_1,\ldots,E_r)]\!]\eta_j\\
&\quad= \textstyle\bigsqcup_{j=0}^{\infty} \eta_j(f)(\mathcal{D}[\![E_1]\!]\eta_j,\ldots,\mathcal{D}[\![E_r]\!]\eta_j) && \text{Axiom 4}\\
&\quad= \textstyle\bigsqcup_{j=0}^{\infty} \bigsqcup_{j'=0}^{\infty} \eta_j(f)(\mathcal{D}[\![E_1]\!]\eta_{j'},\ldots,\mathcal{D}[\![E_r]\!]\eta_{j'}) && \text{mon. of } \eta_j(f)\\
&\quad= \textstyle\bigsqcup_{j=0}^{\infty} \eta_j(f)(\bigsqcup_{j'=0}^{\infty} \mathcal{D}[\![E_1]\!]\eta_{j'},\ldots,\bigsqcup_{j'=0}^{\infty} \mathcal{D}[\![E_r]\!]\eta_{j'}) && \text{cont. of } \eta_j(f)\\
&\quad= \textstyle\bigsqcup_{j=0}^{\infty} \eta_j(f)(\mathcal{D}[\![E_1]\!]\eta,\ldots,\mathcal{D}[\![E_r]\!]\eta) && \text{ind. hyp.}\\
&\quad= \eta(f)(\mathcal{D}[\![E_1]\!]\eta,\ldots,\mathcal{D}[\![E_r]\!]\eta) && \text{prop. of } \sqsubseteq\\
&\quad= \mathcal{D}[\![f(E_1,\ldots,E_r)]\!]\eta && \text{Axiom 4.}\ \square
\end{aligned}$$

Theorem 3-7. Let E be a generalized term and let $\eta \sqsubseteq \eta'$ be environments in which E is meaningful. If η is an environment of monotonic functions, then $\mathcal{D}[\![E]\!]\eta \sqsubseteq \mathcal{D}[\![E]\!]\eta'$.

Proof: Recall the proof of monotonicity of $\lambda\eta.\mathcal{D}[\![E]\!]\eta$ in the first paragraph of the proof of Theorem 3-6. Notice that this proof only depends on η being an environment of monotonic functions. In particular, it does not depend on η' being an environment of continuous, or even monotonic, functions. □

Theorem 3-8. For any program P, the function formed by restricting N_P to environments of continuous functions defined only on F_P is a continuous function.

Proof: We will first show that N_P is a monotonic function when suitably restricted. Let $\eta \sqsubseteq \eta'$ be environments of continuous functions defined only on F_P. Then,

$$\begin{aligned} N_P(\eta)(f_k)(x_1,\ldots,x_{r_k}) &= \mathcal{D}[\![R_k[x_1,\ldots,x_{r_k}]]\!]\eta && \text{def. of } N_P \\ &\sqsubseteq \mathcal{D}[\![R_k[x_1,\ldots,x_{r_k}]]\!]\eta' && \text{Theorem 3-6} \\ &= N_P(\eta')(f_k)(x_1,\ldots,x_{r_k}) && \text{def. of } N_P. \end{aligned}$$

So $N_P(\eta) \sqsubseteq N_P(\eta')$, as needed.

To prove continuity, let $\{\eta_j\}$ be a chain of environments of continuous functions defined only on F_P. Let η be the least upper bound of $\{\eta_j\}$. Then,

$$\begin{aligned} N_P(\eta)(f_k)(x_1,\ldots,x_{r_k}) &= \mathcal{D}[\![R_k[x_1,\ldots,x_{r_k}]]\!]\eta && \text{def. of } N_P \\ &= \bigsqcup_{j=0}^{\infty} \mathcal{D}[\![R_k[x_1,\ldots,x_{r_k}]]\!]\eta_j && \text{Theorem 3-6} \\ &= \bigsqcup_{j=0}^{\infty} N_P(\eta_j)(f_k)(x_1,\ldots,x_{r_k}) && \text{def. of } N_P. \quad \square \end{aligned}$$

Theorem 3-9. Let P be a program, and let $\eta \sqsubseteq \eta'$ be environments defined only on F_P. If η is an environment of monotonic functions, then $N_P(\eta) \sqsubseteq N_P(\eta')$.

Proof: Recall the proof of monotonicity of N_P in the first paragraph of the proof of Theorem 3-8. Notice that if the reference to Theorem 3-6 is changed to reference Theorem 3-7, then this proof depends only on η being an environment of monotonic functions. In particular, it does not depend on η' being an environment of continuous, or even monotonic, functions. □

For any program P, define the environment η_0^P to be undefined for any function variable not declared in F_P, and such that

$$\eta_0^P(f_k) = \lambda x_1,\ldots,x_{r_k}.\bot$$

for any k between 1 and n, where n is the number of declarations in P. For all non-negative integers j define

$$\eta_{j+1}^P = N_P(\eta_j^P).$$

Theorem 3-10. For any program P, any non-negative integer j, and any function variable f_k in F_P, the function $\eta_j^P(f_k)$ is continuous.

Proof: The proof is by induction on j. The function $\eta_0^P(f_k)$ is constant and, therefore, continuous. To prove the induction step, notice that

$$\begin{aligned} \eta_{j+1}^P(f_k) &= N_P(\eta_j^P)(f_k) && \text{def. of } \eta_{j+1} \\ &= \lambda x_1, \ldots, x_{r_k}.\mathcal{D}[\![R_k[x_1, \ldots, x_r]]\!]\eta_j && \text{def. of } N_P. \end{aligned}$$

Therefore, $\eta_{j+1}^P(f_k)$ is continuous by the induction hypothesis and Theorem 3-5. □

Theorem 3-11. For any program P, the set $\{\eta_j^P\}$ forms a chain.

Proof: We prove by induction on j that $\eta_j^P \sqsubseteq \eta_{j+1}^P$ for all j. The base case $\eta_0^P \sqsubseteq \eta_1^P$ holds since η_0^P is the least environment with domain F_P. To prove the induction step, we must show that $\eta_{j+1}^P \sqsubseteq \eta_{j+2}^P$ given the induction hypothesis $\eta_j \sqsubseteq \eta_{j+1}$. But this follows easily from Theorems 3-8 and 3-10. □

Since $\{\eta_j^P\}$ is a chain for all programs P, we can define η^P to be its least upper bound. The least upper bound of a chain of continuous functions is itself a continuous function, so η^P is an environment of continuous functions.

Theorem 3-12. For any program P, the least fixed point of N_P exists and is equal to η^P.

Corollary. For any program P, the least fixed point of the declarations of P exists and is equal to η^P.

Proof: The corollary follows immediately from Theorem 3-4. The first step of the proof of the theorem is to show that η^P is a fixed point of N_P. This follows since,

$$\begin{aligned} N_P(\eta^P) &= \bigsqcup_{j=0}^{\infty} N_P(\eta_j^P) && \text{Theorem 3-8} \\ &= \bigsqcup_{j=0}^{\infty} \eta_{j+1}^P && \text{def. of } \eta_{j+1}^P \\ &= \eta^P && \text{def. of } \eta^P. \end{aligned}$$

To prove that η^P is the least fixed point, let η' be an arbitrary fixed point of N_P. Then, $\eta_0^P \sqsubseteq \eta'$. Also, if $\eta_j^P \sqsubseteq \eta'$, then $\eta_{j+1}^P \sqsubseteq \eta'$ by Theorem 3-9. Therefore, $\eta^P \sqsubseteq \eta'$. □

The environment η^P, which by the previous corollary is the least fixed point of the declarations of the program P, is called the *value environment of* P. An environment η is called a *value environment* if it is the value environment of some program. The value environment of a program P plays an important role in determining the denotation of P, as specified in the following axiom.

Axiom 15. (Program Denotation) If P is a program, then $\mathcal{P}[\![P]\!] = \mathcal{D}[\![E^P]\!]\eta^P$, where E^P is the value term of P, and η^P is the value environment of P.

Since the denotation of any program P is defined without any reference to environments that are not environments of continuous functions, we could have assumed without loss of generality that all environments are of continuous functions. Such an assumption would simply be a matter of convenience, we have shown that it is not necessary for any of the required proofs.

Theorem 3-13. If P_1 and P_2 are two programs such that any declaration in P_1 is included in P_2, then $\eta^{P_1}(f) = \eta^{P_2}(f)$ for any function variable f in F^{P_1}.

Proof: It is sufficient to show that $\eta_j^{P_1}(f) = \eta_j^{P_2}(f)$ for any f in F^{P_1} and any non-negative integer j. This is done using induction on j. The base case of $j = 0$ follows easily from the definition of η_0^P for any program P. To prove the induction step, remember that

$$\eta_{j+1}^{P_1}(f_k) = \lambda x_1, \ldots, x_{r_k}.\mathcal{D}[\![R_k[x_1, \ldots, x_{r_k}]]\!]\eta_j^{P_1}$$

and

$$\eta_{j+1}^{P_2}(f_k) = \lambda x_1, \ldots, x_{r_k}.\mathcal{D}[\![R_k[x_1, \ldots, x_{r_k}]]\!]\eta_j^{P_2},$$

where f_k is in F^{P_1} and

$$f_k(v_1, \ldots, v_{r_k}) \Leftarrow R_k[v_1, \ldots, v_{r_k}]$$

is a declaration in P_1. It follows from the induction hypothesis and Theorem 3-2 that $\eta_{j+1}^{P_1}(f_k) = \eta_{j+1}^{P_2}(f_k)$ for all f_k in F^{P_1}. □

There remains one more axiom to be included in $\mathcal{S}$ and $\mathcal{L}$. This axiom involves the notions of *finite objects* and *infinite objects.* To formalize these notions, we need the following definitions. We write *hd* and *tl* to refer both to function symbols, and to the interpretations of those function symbols.

The set of *extraction functions* is the smallest set of functions from D to D that is closed under composition and contains the identity function, and the functions *hd* and *tl.* The *rank* of an extraction function is defined inductively as follows: the rank of the identity function is 0; and if the rank of e is n, then the rank of $hd \circ e$ and $tl \circ e$ is $n + 1$. If for every non-negative integer n there exists an extraction function e of rank n such that $e(z)$ is a constructed object, then z is an *infinite object,* otherwise it is a *finite object.* The *rank* of an unconstructed object is 0. If z is a finite constructed object, and n is the largest integer such that there exists an extraction function e of rank n with $e(z)$ being a constructed object, then the *rank* of z is $n + 1$.

If $tl^n(z)$ is a constructed object for every non-negative integer n, then z is an *object of infinite length,* otherwise it is an *object of finite length.* The *inclusive length* of an unconstructed object is 1. If z is a constructed object of finite length, and n is the largest integer such that $tl^n(z)$ is a constructed object, then the *inclusive length* of z is $n+1$. If z is an object of finite length with an inclusive length of $n+1$, then the *exclusive length* of z is n. We will see that the informal notion of the *length* of a constructed object may mean either the inclusive length or the exclusive length.

The *hd-rank* of an extraction function is defined inductively as follows: the hd-rank of tl^k is 0 for all non-negative integers k; and if the hd-rank of e is n, then the hd-rank of $tl^k \circ hd \circ e$ is $n+1$ for all non-negative integers k. If for every non-negative integer n there exists an extraction function e with hd-rank n such that $e(z)$ is a constructed object, then z is an *object of infinite depth,* otherwise it is an *object of finite depth.* The *depth* of an unconstructed object is 0. If z is a constructed object of finite depth, and n is the largest integer such that there exists an extraction function e of hd-rank n with $e(z)$ being a constructed object, then the *depth* of z is $n+1$.

Clearly an object x is an infinite object if and only if it is an object of infinite depth or there exists an extraction function e (possibly the identity function) such that $e(x)$ is a object of infinite length.

A function over D is said to be *finiteness preserving* if it maps finite objects to finite objects. An environment η is said to be *finiteness preserving* if $\eta(f)$ is a finiteness preserving function for all function variables f for which $\eta(f)$ is defined.

Theorem 3-14. If P is a program and j is a non-negative integer, then $\mathcal{D}[\![E_P]\!]\eta_j^P$ is a finite object.

Corollary. $\mathcal{P}[\![P]\!]$ is the least upper bound of a chain of finite objects.

Proof: The corollary follows from Theorem 3-6, Axiom 15, and the definition of η^P. Let η be a finiteness preserving environment and let E be a generalized term that is meaningful in η. Also assume that any subterm E' of E that is an object in D is finite. Clearly E has this property if it is a program term, since a program term has no subterms that are objects in D. We will show by induction on the structure of E that $\mathcal{D}[\![E]\!]\eta$ is a finite object. If E is a base term, then $\mathcal{D}[\![E]\!]\eta$ is clearly finite, given our assumptions about E. If the root of E is a function symbol g, then $\mathcal{D}[\![E]\!]\eta$ is a finite object by the induction hypothesis, since $g^{\mathcal{D}}$ is finiteness preserving for any function symbol g. Similar reasoning applies if the root of E is a function variable, since η is finiteness preserving.

It now suffices to show that η_j^P is finiteness preserving for non-negative j. This is done by induction on j. For the base case, η_0^P is clearly finiteness preserving since $\eta_0^P(f_k)$ is finiteness preserving for any f_k in F_P. To prove the induction step, recall that

$$\eta_{j+1}^P(f_k) = \lambda x_1, \ldots, x_{r_k}.\mathcal{D}[\![R_k[x_1, \ldots, x_{r_k}]]\!]\eta_j^P.$$

Combining the induction hypothesis and the result from the previous paragraph shows that $\eta_{j+1}^P(f_k)$ is a finiteness preserving function for all f_k in F_P. Therefore, η_{j+1}^P is finiteness preserving. □

The above theorem shows that the denotation of any program is equal to the least upper bound of some chain of finite objects. Since we are only interested in objects that are the denotation of some program, we may include the following axiom in $\mathcal{S}$ and $\mathcal{L}$.

Axiom 16. (Finitely Approximable) For any object x there exists a chain $\{x_j\}$ of finite objects such that

$$x = \bigsqcup_{j=0}^{\infty} x_j.$$

This completes the axiomization of the strict and non-strict semantics of L. We complete this section with some useful results true in both $\mathcal{S}$ and $\mathcal{L}$.

For any program P define the total function ϕ_P that has as its domain and codomain the set of terms meaningful in η^P, and is such that

$$\phi_P[\![E]\!] = \begin{cases} R_k[E_1, \ldots, E_{r_k}] & \text{if } E = f_k[E_1, \ldots, E_{r_k}]; \\ E & \text{otherwise.} \end{cases}$$

If E has a function variable as its root, then $\phi_P[\![E]\!]$ is the result of applying the copy rule to E. Also define the total function Φ_P with the same domain and codomain such that

$$\Phi_P[\![E]\!] = \begin{cases} \phi_P[\![g(\Phi_P[\![E_1]\!], \ldots, \Phi_P[\![E_r]\!])]\!] & \text{if } E = g(E_1, \ldots, E_r); \\ E & \text{if } E \text{ is a base term;} \end{cases}$$

where g is a function symbol or a function variable. The term $\Phi_P[\![E]\!]$ is the result of applying the copy rule to every function variable instance in E.

An environment η_j^P can be thought of as an approximation to the value environment η^P; the larger j is, the better the approximation. Thus, $\mathcal{D}[\![E_P]\!]\eta_j^P$ can be thought of as an approximation to the denotation of a program P. The following theorem shows that the denotation of P is better approximated by $\mathcal{D}[\![\Phi_P[\![E]\!]]\!]\eta_j^P$ than by $\mathcal{D}[\![E]\!]\eta_j^P$. This result will be used when we discuss the correctness of the implementation of L given in Section 7.

Theorem 3-15. For any program P, variable-free term E, and non-negative integer j,

$$\mathcal{D}[\![\Phi_P[\![E]\!]]\!]\eta_j^P = \mathcal{D}[\![E]\!]\eta_{j+1}^P.$$

Corollary. $\mathcal{D}[\![\Phi_P[\![E]\!]]\!]\eta^P = \mathcal{D}[\![E]\!]\eta^P$.

Proof: The proof is by induction on the structure of E. Theorem clearly holds when E is a base term, since in that case $\Phi_P[\![E]\!] = E$. If the root of E is a function symbol g, then

$$\begin{aligned}
&\mathcal{D}[\![\Phi_P[\![g(E_1, \ldots, E_r)]\!]]\!]\eta_j^P \\
&\quad = \mathcal{D}[\![\phi_P[\![g(\Phi_P[\![E_1]\!], \ldots, \Phi_P[\![E_r]\!])]\!]]\!]\eta_j^P && \text{def. of } \Phi_P \\
&\quad = \mathcal{D}[\![g(\Phi_P[\![E_1]\!], \ldots, \Phi_P[\![E_r]\!])]\!]\eta_j^P && \text{def. of } \phi_P \\
&\quad = \mathcal{D}[\![g(\mathcal{D}[\![\Phi_P[\![E_1]\!]]\!]\eta_j^P, \ldots, \mathcal{D}[\![\Phi_P[\![E_r]\!]]\!]\eta_j^P)]\!]\eta_j^P && \text{Theorem 3-3} \\
&\quad = \mathcal{D}[\![g(\mathcal{D}[\![E_1]\!]\eta_{j+1}^P, \ldots, \mathcal{D}[\![E_r]\!]\eta_{j+1}^P)]\!]\eta_j^P && \text{ind. hyp.} \\
&\quad = \mathcal{D}[\![g(\mathcal{D}[\![E_1]\!]\eta_{j+1}^P, \ldots, \mathcal{D}[\![E_r]\!]\eta_{j+1}^P)]\!]\eta_{j+1}^P && \text{Theorem 3-2} \\
&\quad = \mathcal{D}[\![g(E_1, \ldots, E_r)]\!]\eta_{j+1} && \text{Theorem 3-3.}
\end{aligned}$$

If the root of E is a function variable f not in F_P then the induction step clearly goes through. If the root of E is a function variable f in F_P, then

$$\begin{aligned}
&\mathcal{D}[\![\Phi_P[\![f_k(E_1,\ldots,E_r)]\!]]\!]\eta_j^P \\
&\quad = \mathcal{D}[\![\phi_P[\![f_k(\Phi_P[\![E_1]\!],\ldots,\Phi_P[\![E_r]\!])]\!]]\!]\eta_j^P && \text{def. of } \Phi_P \\
&\quad = \mathcal{D}[\![R_k[\Phi_P[\![E_1]\!],\ldots,\Phi_P[\![E_r]\!]]]\!]\eta_j^P && \text{def. of } \phi_P \\
&\quad = \mathcal{D}[\![R_k[\mathcal{D}[\![\Phi_P[\![E_1]\!]]\!]\eta_j^P,\ldots,\mathcal{D}[\![\Phi_P[\![E_r]\!]]\!]\eta_j^P]]\!]\eta_j^P && \text{Theorem 3-3} \\
&\quad = \mathcal{D}[\![R_k[\mathcal{D}[\![E_1]\!]\eta_{j+1}^P,\ldots,\mathcal{D}[\![E_r]\!]\eta_{j+1}^P]]\!]\eta_j^P && \text{ind. hyp.} \\
&\quad = \eta_{j+1}^P(f_k)(\mathcal{D}[\![E_1]\!]\eta_{j+1}^P,\ldots,\mathcal{D}[\![E_r]\!]\eta_{j+1}^P) && \text{def. of } \eta_{j+1}^P \\
&\quad = \mathcal{D}[\![f_k(E_1,\ldots,E_r)]\!]\eta_{j+1}^P && \text{Axiom 4. } \square
\end{aligned}$$

In a programming language with strict semantics, objects constructed with *cons* are typically called *lists*. However, in $\mathcal{L}$ it is possible to construct objects such as $1:1$ that would not be considered lists. We must formally define which objects we call lists. The only unconstructed object that is a *list* is $\diamondsuit$ (the empty list). A constructed object x is a *list* if and only if for all extraction functions e such that $e(x)$ is a constructed object we have that, if $tl(e(x))$ is unconstructed then it is $\diamondsuit$, and if $hd(e(x))$ is unconstructed then it is an atom or $\diamondsuit$.

Theorem 3-16. If x is a constructed object, then x is a list if and only if $tl(x)$ is a list and $hd(x)$ is an atom or a list.

Proof: To prove the forward implication, assume x is a list. If $tl(x)$ is unconstructed, then it must be $\diamondsuit$ and is, therefore, a list. If $tl(x)$ is constructed, then for all extraction functions e such that $e(tl(x))$ is a constructed object we have that, if $tl(e(tl(x)))$ is unconstructed then it is $\diamondsuit$, and if $hd(e(tl(x)))$ is unconstructed then it is an atom or $\diamondsuit$. Therefore, $tl(x)$ is a list. A similar argument shows that $hd(x)$ is an atom or a list.

To prove the reverse implication, assume $tl(x)$ is a list and $hd(x)$ is an atom or a list. Then, we claim, for all extraction functions e such that $e(x)$ is a constructed object we have that, if $tl(e(x))$ is unconstructed then it is $\diamondsuit$, and if $hd(e(x))$ is unconstructed then it is an atom or $\diamondsuit$. This claim can be verified by considering three cases depending on whether e is the identity function, or is of the form $e' \circ tl$ or $e' \circ hd$. Therefore, x is a list. $\square$

Intuitively, the length of the constructed object $1:1$ is 2, while the length of the list $1:\diamondsuit$ is 1. To match this intuition, the *length* of an object x is its exclusive length if x is a list. Otherwise, the *length* of x is its inclusive length.

We will also be using the notion of a *canonical term*. All base terms are *canonical terms*. A non-base term $E_1 : E_2$ is a *canonical term* if and only if E_1 and E_2 are canonical terms, and $E_1 : E_2$ is equivalent to some constructed object. Intuitively, a canonical term is a term that is fully reduced or simplified. Notice that a canonical term is necessarily a simple term, so its denotation is independent of environment. An object x of D is a *canonical object* if and only if it is equivalent

to some canonical term. Thus, a canonical object is an object that is equivalent to some fully reduced term. It is clear that any canonical object is a finite object. We also have the following result.

Theorem 3-17. If E and E' are distinct canonical terms, then $E \not\sqsubseteq E'$.

Proof: The proof is by induction of the maximum of the ranks of the two objects denoted by E and E'. If either E or E' denote an unconstructed object, then the theorem clearly holds. If E and E' both denote constructed objects, then there exist canonical terms E_1, E_2, E_1', and E_2' such that $E = E_1 : E_2$ and $E' = E_1' : E_2'$. By the induction hypothesis, either $E_1 \not\sqsubseteq E_1'$, or $E_2 \not\sqsubseteq E_2'$, or both. Therefore, $E \not\sqsubseteq E'$. □

Two important facts follow from this theorem. The first is that any two distinct canonical terms denote distinct objects. The second is that any two distinct canonical objects are not comparable.

4. Some Properties of $\mathcal{S}$

The axiomizations $\mathcal{S}$ and $\mathcal{L}$ differ only in Axiom 5, the specification of *cons*. However, this difference leads to several significant differences in the properties of $\mathcal{S}$ and $\mathcal{L}$. In this section we prove several properties of $\mathcal{S}$, which can be compared to the properties of $\mathcal{L}$ discussed in the following section. Unless otherwise noted, the statements made in this section apply only to $\mathcal{S}$.

One of the most important differences between our strict and non-strict semantics is that non-strict semantics allows for infinite objects while strict semantics does not. We prove in Theorem 4-2 that in $\mathcal{S}$ all objects are finite. That proof requires the following theorem.

Theorem 4-1. If x and y are finite objects, then

$$x \sqsubseteq y \iff (x \equiv \bot \vee x \equiv y).$$

Proof: The reverse implication follows directly, we need only show the forward implication. The forward implication also follows directly in the cases where either x or y is an unconstructed object. So, all that remains is the case where x and y are both finite constructed objects, which we can prove by induction on the maximum of the ranks of x and y.

$$\begin{aligned}
x \sqsubseteq y &\Rightarrow hd(x) \sqsubseteq hd(y) \wedge tl(x) \sqsubseteq tl(y) && \text{prop. of } \sqsubseteq \\
&\Rightarrow (hd(x) \equiv \bot \vee hd(x) \equiv hd(y)) \wedge && \text{ind. hyp.} \\
&\quad\ (tl(x) \equiv \bot \vee tl(x) \equiv tl(y)) && \\
&\Rightarrow hd(x) \equiv hd(y) \wedge tl(x) \equiv tl(y) && \text{see below} \\
&\Rightarrow hd(x) : tl(x) \equiv hd(y) : tl(y) && \text{substitution} \\
&\Rightarrow x \equiv y && \text{prop. of } hd \text{ and } tl.
\end{aligned}$$

The third step is justified since if x is a constructed object, then $hd(x) \not\equiv \bot$ and $tl(x) \not\equiv \bot$. □

Theorem 4-2. All objects are finite objects.

Corollary. If x and y are arbitrary objects, then

$$x \sqsubseteq y \iff (x \equiv \perp \vee x \equiv y).$$

Proof: The corollary follows immediately from Theorem 4-1. To prove the theorem, consider an arbitrary object x. By the Approximability Axiom there exists a chain $\{x_j\}$ of finite objects such that x is the least upper bound of x_j. If $x \equiv \perp$ then x is a finite object. If $x \not\equiv \perp$ then there exists an n such that $x_n \not\equiv \perp$. Since $\{x_j\}$ is a chain, it follows that $x_n \sqsubseteq x_m$ for all m such that $n \leq m$. So $x_n \equiv x_m$ for all such m by Theorem 4-1, since all the x_j are finite objects. Therefore $x \equiv x_n$, so x is finite. □

The corollary to Theorem 4-2 states that the domain D is a *flat* domain. We will see that in $\mathcal{L}$ the partial ordering over D has a much more complicated structure.

Any strict semantics for lists should have the property that all constructed objects are lists. We prove that $\mathcal{S}$ has this property in the following theorem.

Theorem 4-3. Every constructed object is a list.

Proof: Consider an arbitrary object x. The theorem may be proved using induction on the rank of x since all objects are finite. If the rank of x is 0, then x is not a constructed object, so the theorem holds trivially.

For the induction step, assume the rank of x is greater than 0. Since x is a constructed object, $tl(x)$ must be a constructed object or $\diamondsuit$, and $hd(x)$ must be an atom, a constructed object, or $\diamondsuit$. Therefore, by the induction hypothesis, $tl(x)$ must be a list and $hd(x)$ must be an atom or a list. Therefore, by Theorem 3-16, x is a list. □

In order to prove useful properties of programs, it is necessary to be able show whether or not two given lists are equal. The normal method for doing this is to prove that each of the corresponding items in the two lists are equal. The following theorem makes available a more general proof method.

Theorem 4-4. If x and y are objects, then $x \sqsubseteq y$ if and only if $e(x) \sqsubseteq e(y)$ for every extraction function e such that either $e(x)$ is unconstructed or $e(y)$ is unconstructed.

Proof: The forward implication is true since all extraction functions are monotonic. The proof of the reverse implication is by induction on the rank of x. If x is an unconstructed object, then it follows trivially that $x \sqsubseteq y$. Consider the case in which x is a constructed object. In this case y is a constructed object since $e(x) \sqsubseteq e(y)$ for all extraction functions such that $e(y)$ is unconstructed. By the induction hypothesis, $hd(x) \sqsubseteq hd(y)$ and $tl(x) \sqsubseteq tl(y)$. Therefore, $x \sqsubseteq y$. □

The intuitive interpretation of the following theorem is that every object other than $\perp$ is equivalent to some fully reduced term.

Theorem 4-5. Every object other than $\perp$ is a canonical object.

Proof: Consider an arbitrary object z distinct from $\perp$. The proof is by induction on the rank of z. If z is an unconstructed object then it is clearly a canonical object. If z is a constructed object, then there exist x and y distinct from $\perp$ such that $z \equiv x : y$. By the induction hypothesis, there exist canonical terms E and E' such that $x \equiv E$ and $y \equiv E'$. Therefore, z is a canonical object since $z \equiv E : E'$ and $E : E'$ is a canonical term. □

The above results show that $\mathcal{S}$ has many of the properties our intuition would expect for a strict semantics of list. In Section 8 we show that these properties are sufficient to prove the correctness of several example programs.

5. Some Properties of $\mathcal{L}$

In this section we prove several properties of $\mathcal{L}$, which can be compared to the properties of $\mathcal{S}$ discussed in the previous section. Unless otherwise noted, the statements made in this section apply only to $\mathcal{L}$.

Let P be a program that includes the declaration

$$ones \Leftarrow 1 : ones$$

(we omit parenthesis rather than write $ones()$ since the function variable *ones* is declared to be of arity zero). Consider the denotation of the term *ones* in the value environment of P. Notice that the denotation is independent of any other declarations in P by Theorem 3-13. In the future we will simply refer to this object as *ones* and the declaration above will be implicit. The object *ones* is important because, as is proven in the following theorem, it is a simple example of an infinite list.

Theorem 5-1. The object *ones* is a list of infinite length.

Proof: Clearly, *ones* is a constructed object. Also, $tl(ones) \equiv ones$. It follows by induction that $tl^k(ones) \equiv ones$ for all non-negative integers k. Therefore, *ones* is an object of infinite length.

To see that *ones* is a list, notice that any extraction function e such that $e(ones)$ is constructed is of the form tl^n; this is because $tl^n(ones) \equiv ones$ and $hd(tl^n(ones)) \equiv 1$. For any e of this form, $tl(e(ones))$ is a constructed object (the infinite object *ones*), and $hd(e(ones))$ is an atom (the atom 1). Therefore, *ones* is a list. □

It is not immediately clear that the ordering $\sqsubseteq$ is completely determined by the axioms of $\mathcal{L}$. It is clearly determined for finite objects in D since we can prove by induction over their structure whether or not two finite objects are comparable. But such structural induction does not apply to infinite objects, so we most prove by other means that $\sqsubseteq$ is completely determined. This is done with the following two theorems.

Theorem 5-2. If x is a finite object and y is the least upper bound of a chain $\{y_j\}$, then

$$x \sqsubseteq y \iff \exists j(x \sqsubseteq y_j).$$

Proof: The reverse implication is clear by the definition of the least upper bound. To prove the forward implication, assume $x \sqsubseteq y$. We prove by induction on the rank of x that $\exists j(x \sqsubseteq y_j)$.

The proof of the base case has two cases depending on whether or not x is equivalent to $\perp$. The $x \equiv \perp$ case is trivial. If x is an unconstructed object other than $\perp$, then $x \equiv y$, which implies that y as an unconstructed object other than $\perp$. Thus, there exists a j such that $y \equiv y_j$, which implies that $x \equiv y_j$.

For the induction step, assume that x is a constructed object, which implies that y is a constructed object. Therefore, there exists a k such that y_k is a constructed object. By the induction hypothesis there exists a $j_0 \geq k$ and a $j_1 \geq k$ such that $hd(x) \sqsubseteq hd(y_{j_0})$ and $tl(x) \sqsubseteq tl(y_{j_1})$. Let j be the maximum of j_0 and j_1. Then,

$$\begin{aligned} x &\equiv hd(x) : tl(x) && \text{prop. of } hd \text{ and } tl \\ &\sqsubseteq hd(y_j) : tl(y_j) && \text{prop. of } j \\ &\equiv y_j && \text{prop. of } hd \text{ and } tl. \quad \square \end{aligned}$$

Theorem 5-3. Let $\{x_j\}$ and $\{y_j\}$ be chains with least upper bounds of x and y, respectively. If all of the x_j are finite, then

$$x \sqsubseteq y \iff \forall j \exists k(x_j \sqsubseteq y_k).$$

Proof: The theorem follows from the following logical equivalences:

$$\begin{aligned} x \sqsubseteq y &\iff \forall j(x_j \sqsubseteq y) && \text{def. of } \bigsqcup \\ &\iff \forall j \exists k(x_j \sqsubseteq y_k) && \text{Theorem 5-2.} \quad \square \end{aligned}$$

Let x and y be arbitrary objects. By the finite approximation axiom there are chains of finite objects that have x and y as their least upper bounds. Thus, by Theorem 5-3, the assertion $x \sqsubseteq y$ is true if and only if a statement involving only comparisions of finite objects is true. Therefore, since $\sqsubseteq$ is completely determined for finite objects, $\sqsubseteq$ is completely determined for all objects.

Theorem 4-4 is a useful theorem for doing correctness proofs of programs with semantics given by $\mathcal{S}$. A similar theorem holds for $\mathcal{L}$ as well.

Theorem 5-4. If x and y are objects, then $x \sqsubseteq y$ if and only if $e(x) \sqsubseteq e(y)$ for every extraction function e such that either $e(x)$ is unconstructed or $e(y)$ is unconstructed.

Proof: The forward implication is true since all extraction functions are monotonic. If x is a finite object, then the proof of the reverse implication is by induction on

the rank of x just as in the proof of Theorem 4-4. Notice that this proof does not require that y be finite.

To prove the theorem for infinite x, let $\{x_j\}$ be a chain of finite objects with x as its least upper bound. Consider an arbitrary x_j. Also, consider the set F of all extraction functions e such that either $e(x_j)$ is unconstructed or $e(y)$ is unconstructed. It can be shown that $e(x_j) \sqsubseteq e(y)$ for all e in F by considering the following three cases.

If $e(y)$ is unconstructed then $e(x) \sqsubseteq e(y)$, which implies $e(x_j) \sqsubseteq e(y)$. If $e(x_j)$ and $e(x)$ are unconstructed, then $e(x) \sqsubseteq e(y)$; which again implies $e(x_j) \sqsubseteq e(y)$. If $e(x_j)$ is unconstructed and $e(x)$ is constructed then $e(x_j) \equiv \perp$, which implies $e(x_j) \sqsubseteq e(y)$.

Thus, $x_j \sqsubseteq y$ since x_j is finite and we have already shown that the theorem holds for finite x. Since x_j was chosen arbitrarily, it follows that $x \sqsubseteq y$ by the definition of least upper bound. □

We have seen that the semantics given by $\mathcal{L}$ allow the construction of objects such as $1:1$ that are not lists. These semantics also allow the construction of another class of unusual objects, some of which are lists: the *unproductive* objects. The *completely unproductive list* is the list x such that $e(x)$ is a constructed object for all extraction functions e. There is only one such list since if x and y are such that $e(x)$ and $e(y)$ are constructed for all extraction functions e, then $x \equiv y$ by two applications of Theorem 5-4. An object x is *unproductive* if $e(x)$ is the completely unproductive list for some extraction function e.

In Theorem 4-5 we saw that in $\mathcal{S}$, every object other than $\perp$ is equivalent to some fully reduced term. In $\mathcal{L}$ there are many objects other than $\perp$ that are not equivalent to some fully reduced term, as shown in the following theorem.

Theorem 5-5. An object x is canonical if and only if it is finite and $e(x) \not\equiv \perp$ for all extraction function e.

Proof: If x is infinite, then both sides of the if and only if are false, so the theorem holds in this case. The proof of the theorem for finite x is by induction on the rank of x. The theorem holds if x is unconstructed since there is a base term equivalent to x if and only if $x \not\equiv \perp$, and all base terms are canonical terms.

If x is constructed, then by the induction hypothesis the theorem holds for $hd(x)$ and $tl(x)$. It is then simple to verify that the theorem holds for x by seperately checking the forward and reverse implications. □

Theorem 5-6. If y is a canonical object, then there exists only a finite number of distinct x such that $x \sqsubseteq y$.

Proof: The proof is by induction on the rank of y. If y is unconstructed then $x \equiv \perp$ and $x \equiv y$ are the only x such that $x \sqsubseteq y$. If y is constructed, then by the induction hypothesis there is only a finite number m of x such that $x \sqsubseteq hd(y)$ and only a finite number n of x such that $x \sqsubseteq tl(y)$. Therefore, there is only a finite number, namely $m \times n + 1$, of x such that $x \sqsubseteq y$. □

In both $\mathcal{S}$ and $\mathcal{L}$ an object x is canonical if and only there exists no $y \not\equiv x$ such that $x \sqsubseteq y$, and there exists only a finite number of distinct y such that $y \sqsubseteq x$. This could be used as an alternative definition of canonical objects.

6. A Model of $\mathcal{L}$

In this section we prove that there exists of a model of $\mathcal{L}$, and that any two models of $\mathcal{L}$ are isomorphic. Defining a model of $\mathcal{L}$ requires giving a domain D with partial ordering $\sqsubseteq^D$, and defining semantic functions $\mathcal{D}$ and $\mathcal{P}$, that together satisfy the axioms given in Section 3. We do not give a complete specification of a model, but only specify a part of a model sufficient to demonstrate that a complete model exists (see [2] for an inverse limit construction of a domain of infinite lists, though this domain does not satisfy $\mathcal{L}$). Specifically, we define a domain D with partial ordering $\sqsubseteq^D$, and also partially specify a semantic function $\mathcal{D}$ (by giving interpretations for the function symbols *cons*, *hd* and *tl*), that satisfy the axioms of $\mathcal{L}$ listed below.

Axiom 1. (Semantic Classes) The objects of D are divided into two disjoint sets, the *unconstructed objects* and the *constructed objects*. The unconstructed objects are $\perp^D$, $\mathcal{E}^D$, $\Diamond^D$ and the members of the set of atoms A.

Axiom 2. (Base Terms) If E is a variable-free generalized base term, then for any environment η

$$\mathcal{D}[\![E]\!]\eta = \begin{cases} \mathcal{E}^D & \text{if } E = \mathcal{E}; \\ \Diamond^D & \text{if } E = \Diamond; \\ \mathcal{A}[\![E]\!] & \text{if } E \text{ is an atomic base term;} \\ E & \text{if } E \text{ is an object in } D. \end{cases}$$

Axiom 3. (Function Symbols) Let g be a function symbol of arity r. There exists a function $g^{\mathcal{D}}$ (the *interpretation of g with respect to $\mathcal{D}$*) from D^r to D such that

$$\mathcal{D}[\![g(E_1, \ldots, E_k)]\!]\eta = g^{\mathcal{D}}(\mathcal{D}[\![E_1]\!]\eta, \ldots, \mathcal{D}[\![E_k]\!]\eta)$$

if $k = r$ and each of the $\mathcal{D}[\![E_i]\!]\eta$ is defined. Otherwise, $\mathcal{D}[\![g(E_1, \ldots, E_k)]\!]\eta$ is undefined.

Axiom 5 of $\mathcal{L}$. (*cons*) For all x and y in D, the generalized term $x : y$ is equivalent to a constructed object.

Axiom 6. (Constructability) For all constructed objects z, there exist unique x and y such that $z \equiv x : y$.

Axiom 7. (*hd* **and** *tl*) For all z in D, if $x : y$ is equivalent to a constructed object, then

$$hd(z) \equiv \begin{cases} \perp & \text{if } z \equiv \perp; \\ \mathcal{E} & \text{if } z \equiv \Diamond \text{ or } z \equiv \mathcal{E}; \\ \mathcal{E} & \text{if } z \text{ is an atom;} \\ x & \text{if } z \equiv x : y; \end{cases} \qquad tl(z) \equiv \begin{cases} \perp & \text{if } z \equiv \perp; \\ \mathcal{E} & \text{if } z = \Diamond \text{ or } z \equiv \mathcal{E}; \\ \mathcal{E} & \text{if } z \text{ is an atom;} \\ y & \text{if } z \equiv x : y. \end{cases}$$

Axiom 13. ($\sqsubseteq$)

a) For all x, $x \sqsubseteq x$. (reflexivity)

b) For all x and y, if $x \sqsubseteq y$ and $y \sqsubseteq x$, then $x \equiv y$. (antisymmetry)

c) For all x, y, and z, if $x \sqsubseteq y$ and $y \sqsubseteq z$, then $x \sqsubseteq z$. (transitivity)

d) If x or y is an unconstructed object, then $x \sqsubseteq y$ if and only if $x \equiv \perp$ or $x \equiv y$.

e) If x and y are constructed objects, then $x \sqsubseteq y$ if and only if $hd(x) \sqsubseteq hd(y)$ and $tl(x) \sqsubseteq tl(y)$.

Axiom 14. (Closure) For all chains $\{x_j\}$ of objects in D, there exists an object x such that

$$x = \bigsqcup_{j=0}^{\infty} x_j.$$

Axiom 16. (Finitely Approximable) For any object x there exists a chain $\{x_j\}$ of finite objects such that

$$x = \bigsqcup_{j=0}^{\infty} x_j.$$

The remaining axioms of $\mathcal{L}$ in Section 3 are all in the form of definitions. Therefore, if our proposed partially specified model does in fact satisfy the axioms listed above, then it can be extended (by completing the definitions of $\mathcal{D}$ and $\mathcal{P}$) in exactly one way to satisfy all the remaining axioms of $\mathcal{L}$. Thus, our partially specified model is sufficient to demonstrate the existence of a model of $\mathcal{L}$.

Let A be the set of atoms, i.e., the integers together with the two truth values. Define the set

$$M = A \cup \{\perp^M, \Diamond^M \mathcal{E}^M, \gamma\}.$$

Thus, M contains elements corresponding to each unconstructed object in the domain D, and also contains an additional element γ, the purpose of which will become clear. We define a partial ordering $\sqsubseteq^M$ over M such that

$$(u \sqsubseteq^M v) \iff (u = \perp^M) \vee (u = v).$$

We can use trees with nodes labeled by elements of M to represent finite objects, as in Figure 1. Such trees are called *full binary trees* because each node has either zero or two children. We call a full binary tree *complete* if all the leaf nodes are the same distance from the root, where the distance of a node from the root is the number of edges in the path from the root to that node. We can transform trees such as that in Figure 1 into complete binary trees by adding nodes as in Figure 2. Nodes added as children of a node labeled with $\perp^M$ are also labeled with $\perp^M$, children of nodes labeled with anything but $\perp^M$ or γ are labeled with $\mathcal{E}^M$. No leaf node is ever labeled with γ, so that case need not be considered.

The tree in Figure 2 was formed by adding the minimum number of nodes necessary to form a complete binary tree. It is possible to add more nodes and still

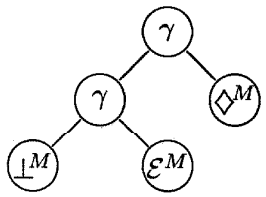

Figure 1: Full binary tree representation of $(\bot : \mathcal{E}) : \Diamond$.

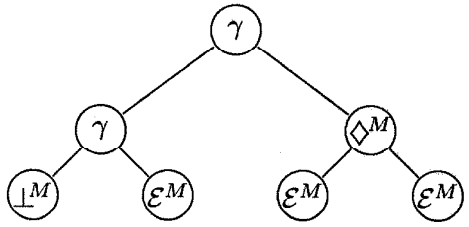

Figure 2: Complete binary tree representation of $(\bot : \mathcal{E}) : \Diamond$.

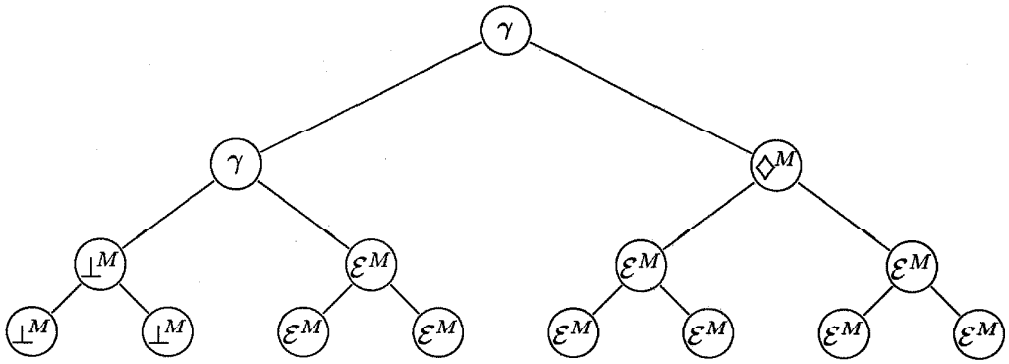

Figure 3: Another complete binary tree representation of $(\bot : \mathcal{E}) : \Diamond$.

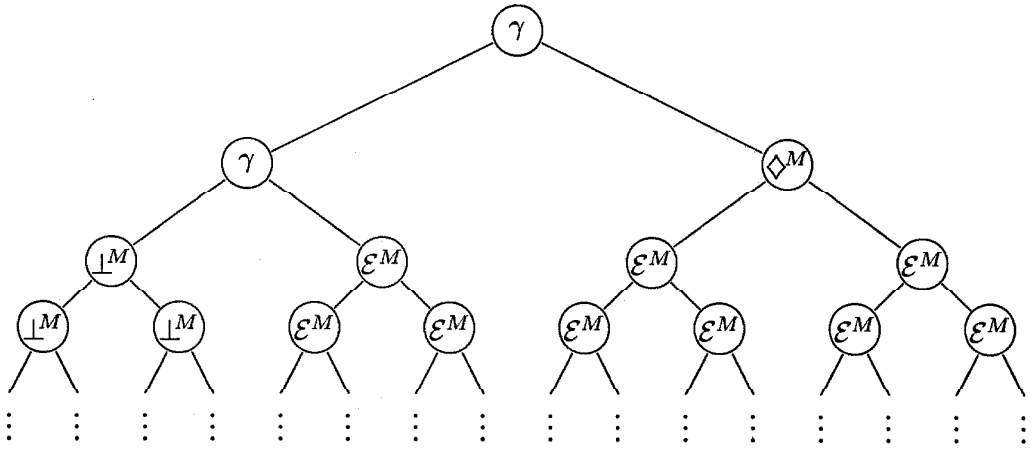

Figure 4: Infinite complete binary tree representation of $(\bot : \mathcal{E}) : \Diamond$.

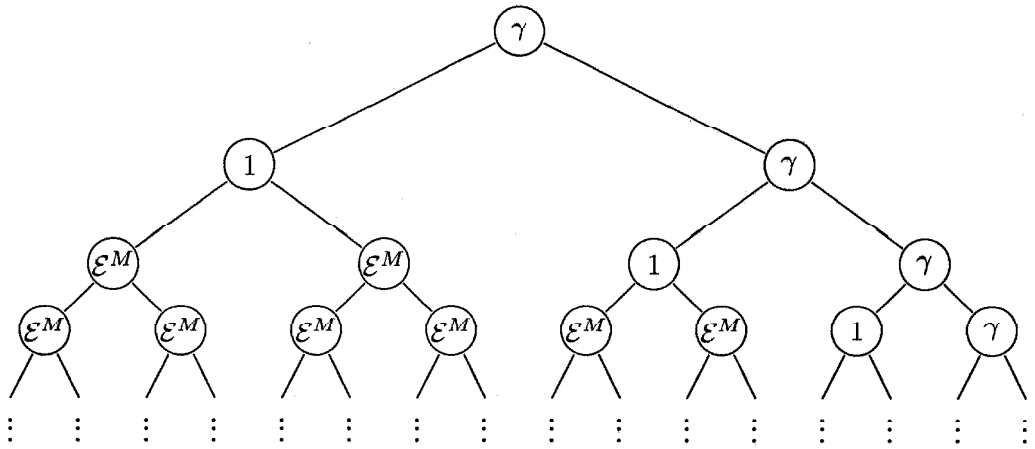

Figure 5: Infinite complete binary tree representation of *ones*.

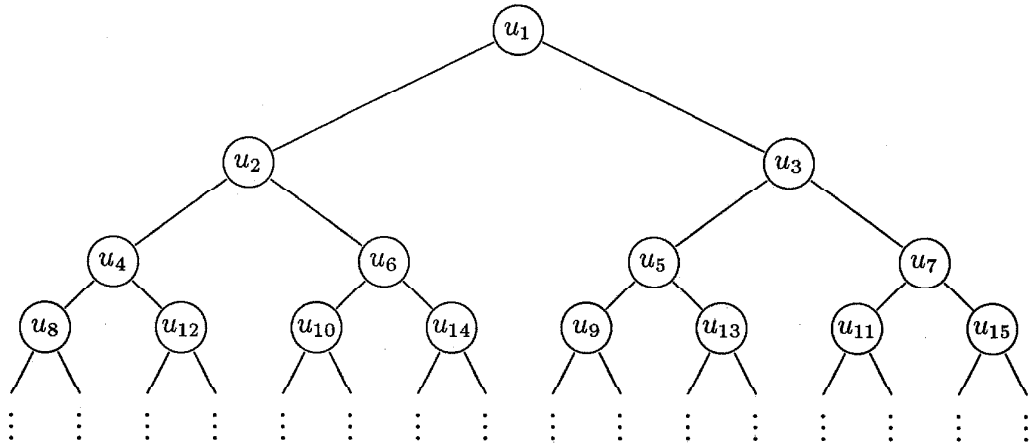

Figure 6: Infinite complete binary tree represented by $\langle u \rangle_{n=1}^{\infty}$.

form a complete binary tree that represents the same finite object, as in Figure 3. In fact, one could imagine forming an infinite complete binary tree in this way, as in Figure 4. Thus, the tree in Figure 4 is an example of an infinite tree used to represent a finite object. Such infinite trees can also be used to represent infinite objects as in Figure 5.

Such a tree can in turn be represented by an infinite sequence of elements of M, i.e., by elements of the set

$$U = \{\langle u_n \rangle_{n=1}^{\infty} \mid \forall n[u_n \in M]\}.$$

We require that elements of such a sequence correspond to the nodes of an infinite complete binary tree as shown in Figure 6. We will see that this correspondence simplifies defining the interpretation of *cons*.

If a particular tree corresponds to the sequence $\langle u_n \rangle_{n=1}^{\infty}$, then the left and right child of u_n are $u_{n+\rho(n)}$ and $u_{n+2\rho(n)}$, respectively, where

$$\rho(n) = 2^{\lfloor log_2(n) \rfloor}.$$

If $\langle u_n \rangle_{n=1}^{\infty}$ corresponds to a complete binary tree formed as informally described earlier, then $Q_{\perp}(\langle u_n \rangle_{n=1}^{\infty})$ and $Q_{\mathcal{E}}(\langle u_n \rangle_{n=1}^{\infty})$ hold, where $Q_{\perp}(\langle u_n \rangle_{n=1}^{\infty})$ is defined as

$$\forall n\Big[(u_n = \perp^M) \Rightarrow ((u_{n+\rho(n)} = \perp^M) \wedge (u_{n+2\rho(n)} = \perp^M))\Big],$$

and $Q_{\mathcal{E}}(\langle u_n \rangle_{n=1}^{\infty})$ is defined as

$$\forall n\Big[((u_n \neq \perp^M) \wedge (u_n \neq \gamma)) \Rightarrow ((u_{n+\rho(n)} = \mathcal{E}^M) \wedge (u_{n+2\rho(n)} = \mathcal{E}^M))\Big].$$

This motivates defining the domain D of our model of $\mathcal{L}$ by

$$D = \{x \mid x \in U \wedge Q_{\perp}(x) \wedge Q_{\mathcal{E}}(x)\}.$$

The partial ordering over M is extended to form the partial ordering $\sqsubseteq^D$ over D such that

$$\langle u_n \rangle_{n=1}^{\infty} \sqsubseteq^D \langle u_n \rangle_{n=1}^{\infty} \iff \forall n[u_n \sqsubseteq^M v_n].$$

An object $x = \langle u_n \rangle_{n=1}^{\infty}$ is defined to be a constructed object if $u_1 = \gamma$, otherwise it is an unconstructed object.

Next we define functions $\mathcal{C} : D \times D \mapsto D$ and $\mathcal{H}, \mathcal{T} : D \mapsto D$ that serve as the interpretations of *cons*, *hd* and *tl*, respectively. The function $\mathcal{C}$ is given by

$$\mathcal{C}(\langle u_n \rangle_{n=1}^{\infty}, \langle v_n \rangle_{n=1}^{\infty}) = \langle w_n \rangle_{n=1}^{\infty},$$

where

$$w_n = \begin{cases} \gamma & \text{if } n = 1; \\ u_{\frac{n}{2}} & \text{if } n > 1 \text{ and } n \text{ is even;} \\ v_{\frac{n-1}{2}} & \text{if } n > 1 \text{ and } n \text{ is odd.} \end{cases}$$

The functions $\mathcal{H}$ and $\mathcal{T}$ are given by

$$\mathcal{H}(\langle u_n \rangle_{n=1}^{\infty}) = \langle u_{2n} \rangle_{n=1}^{\infty},$$
$$\mathcal{T}(\langle u_n \rangle_{n=1}^{\infty}) = \langle u_{2n+1} \rangle_{n=1}^{\infty}.$$

Clearly the above definitions make sense if they are viewed as defining functions over U, but it is a more difficult task to prove that these definitions make sense for functions over D. This task is carried out in the proof of the following theorem.

Theorem 6-1. The functions $\mathcal{C}$, $\mathcal{H}$, and $\mathcal{T}$ are well-defined

Proof: Let $\langle u_n\rangle_{n=1}^{\infty}$ and $\langle v_n\rangle_{n=1}^{\infty}$ be elements of D. This implies that $Q_\perp(\langle u_n\rangle_{n=1}^{\infty})$, $Q_\mathcal{E}(\langle u_n\rangle_{n=1}^{\infty})$, $Q_\perp(\langle v_n\rangle_{n=1}^{\infty})$, and $Q_\mathcal{E}(\langle v_n\rangle_{n=1}^{\infty})$ hold. Let $\langle w_n\rangle_{n=1}^{\infty}$ be defined in terms of $\langle u_n\rangle_{n=1}^{\infty}$ and $\langle v_n\rangle_{n=1}^{\infty}$ as in the definition of $\mathcal{C}$. We must show that the following propositions hold: $Q_\perp(\langle w_n\rangle_{n=1}^{\infty})$, $Q_\mathcal{E}(\langle w_n\rangle_{n=1}^{\infty})$, $Q_\perp(\langle u_{2n}\rangle_{n=1}^{\infty})$, $Q_\mathcal{E}(\langle u_{2n}\rangle_{n=1}^{\infty})$, $Q_\perp(\langle u_{2n+1}\rangle_{n=1}^{\infty})$, and $Q_\mathcal{E}(\langle u_{2n+1}\rangle_{n=1}^{\infty})$. We will actually prove only those propositions concerning $Q_\perp$, the proofs for $Q_\mathcal{E}$ are analogous.

To prove $Q_\perp(\langle w_n\rangle_{n=1}^{\infty})$ we must consider three cases corresponding to the cases in the definition of w_n. The $n = 1$ case is trivial since $w_1 = \gamma$. If $n > 1$ and n is even, then

$$\begin{aligned}
w_n = \perp &\Rightarrow u_{\frac{n}{2}} = \perp && \text{def. of } w_n\\
&\Rightarrow (u_{\frac{n}{2}+\rho(\frac{n}{2})} = \perp) \wedge (u_{\frac{n}{2}+2\rho(\frac{n}{2})} = \perp) && Q_\perp(\langle u_n\rangle_{n=1}^{\infty})\\
&\Rightarrow (u_{\frac{n+\rho(n)}{2}} = \perp) \wedge (u_{\frac{n+2\rho(n)}{2}} = \perp) && \text{prop. of } \rho\\
&\Rightarrow (w_{n+\rho(n)} = \perp) \wedge (w_{n+2\rho(n)} = \perp) && \text{def. of } w_n.
\end{aligned}$$

If $n > 1$ and n is odd, then

$$\begin{aligned}
w_n = \perp &\Rightarrow v_{\frac{n-1}{2}} = \perp && \text{def. of } w_n\\
&\Rightarrow (v_{\frac{n-1}{2}+\rho(\frac{n-1}{2})} = \perp) \wedge (v_{\frac{n-1}{2}+2\rho(\frac{n-1}{2})} = \perp) && Q_\perp(\langle v_n\rangle_{n=1}^{\infty})\\
&\Rightarrow (v_{\frac{n+\rho(n-1)-1}{2}} = \perp) \wedge (v_{\frac{n+2\rho(n-1)-1}{2}} = \perp) && \text{prop. of } \rho\\
&\Rightarrow (v_{\frac{n+\rho(n)-1}{2}} = \perp) \wedge (v_{\frac{n+2\rho(n)-1}{2}} = \perp) && \text{prop. of } \rho\\
&\Rightarrow (w_{n+\rho(n)} = \perp) \wedge (w_{n+2\rho(n)} = \perp) && \text{def. of } w_n.
\end{aligned}$$

The proposition $Q_\perp(\langle u_{2n}\rangle_{n=1}^{\infty})$ follows from

$$\begin{aligned}
u_{2n} = \perp &\Rightarrow (u_{2n+\rho(2n)} = \perp) \wedge (u_{2n+2\rho(2n)} = \perp) && Q_\perp(\langle u_n\rangle_{n=1}^{\infty})\\
&\Rightarrow (u_{2(n+\rho(n))} = \perp) \wedge (u_{2(n+2\rho(n))} = \perp) && \text{prop. of } \rho.
\end{aligned}$$

Finally, the proposition $Q_\perp(\langle u_{2n+1}\rangle_{n-1}^{\infty})$ follows from

$$\begin{aligned}
u_{2n+1} = \perp &\Rightarrow (u_{2n+1+\rho(2n+1)} = \perp) \wedge (u_{2n+1+2\rho(2n+1)} = \perp) && Q_\perp(\langle u_n\rangle_{n=1}^{\infty})\\
&\Rightarrow (u_{2n+1+\rho(2n)} = \perp) \wedge (u_{2n+1+2\rho(2n)} = \perp) && \text{prop. of } \rho\\
&\Rightarrow (u_{2(n+\rho(n))+1} = \perp) \wedge (u_{2(n+2\rho(n))+1} = \perp) && \text{prop. of } \rho. \quad \square
\end{aligned}$$

We have now completed enough of our partial specification of a model of $\mathcal{L}$ to begin demonstrating that the model satisfies the axioms of $\mathcal{L}$ listed at the beginning of this section. Axiom 1 requires that D contain two semantic classes, the constructed objects and the unconstructed objects. We have already specified that an object $x = \langle u_n\rangle_{n=1}^{\infty}$ is a constructed object if $u_1 = \gamma$, otherwise it is an unconstructed object. The axiom also specifies the class of unconstructed objects to be $\perp^D$, $\mathcal{E}^D$, $\Diamond^D$, and the members of the set of atoms A. We define each of these to be the domain element $\langle u_n\rangle_{n=1}^{\infty}$ where, respectively, $u_1 = \perp^M$, $u_1 = \mathcal{E}^M$, $u_1 = \Diamond^M$,

and $u_1 = a$ for some a in A. Notice that these define a unique $\langle u_n \rangle_{n=1}^{\infty}$ in each case since the propositions $Q_{\perp}(\langle u_n \rangle_{n=1}^{\infty})$ and $Q_{\mathcal{E}}(\langle u_n \rangle_{n=1}^{\infty})$ hold.

Axioms 2 and 3 give properties of the semantic function $\mathcal{D}$ in terms of the interpretations $g^{\mathcal{D}}$ of each function symbol g in L. It is clear that for any set of interpretations there exists a $\mathcal{D}$ that satisfies Axioms 2 and 3. We require that $\mathcal{C}$, $\mathcal{H}$ and $\mathcal{T}$ be the interpretations of the function symbols *cons*, *hd* and *tl*, respectively. Axiom 5 requires that $x : y$ be equivalent to a constructed object for any x and y in D. This clearly holds in our model by the definition of $\mathcal{C}$.

Our model satisfies Axiom 6 since if z constructed object in D, then $x = \mathcal{H}(z)$ and $y = \mathcal{T}(z)$ are the only objects such that $z = \mathcal{C}(x, y)$. This fact is also used to verify that $\mathcal{H}$ and $\mathcal{T}$ satisfy Axiom 7. It is also easy to verify that $\sqsubseteq^D$ satisfies Axiom 13.

To verify that our model satisfies Axiom 14, first notice that any chain of elements of M has a least upper bound in M. This fact, together with the following theorem, shows that Axiom 14 is satisfied.

Theorem 6-2. For any chain $\{\langle u_{k,n} \rangle_{n=1}^{\infty}\}$ in D,

$$\bigsqcup_{k=0}^{\infty} (\langle u_{k,n} \rangle_{n=1}^{\infty}) = \Big\langle \bigsqcup_{k=0}^{\infty} u_{k,n} \Big\rangle_{n=1}^{\infty}.$$

Proof: The theorem is proved by showing that the following chain of logical equivalences holds for arbitrary $\langle v_n \rangle_{n=1}^{\infty}$ in D,

$$\begin{aligned}
\forall k[\langle u_{k,n} \rangle_{n=1}^{\infty} \sqsubseteq \langle v_n \rangle_{n=1}^{\infty}] &\iff \forall k \forall n[u_{k,n} \sqsubseteq v_n] && \text{Def. of } \sqsubseteq^D \\
&\iff \forall n\Big[\bigsqcup_{k=0}^{\infty} u_{k,n} \sqsubseteq v_n\Big] && \text{Def. of } \bigsqcup \\
&\iff \Big\langle \bigsqcup_{k=0}^{\infty} u_{k,n} \Big\rangle_{n=1}^{\infty} \sqsubseteq \langle v_n \rangle_{n=1}^{\infty} && \text{Def. of } \sqsubseteq^D. \quad \square
\end{aligned}$$

The last axiom to verify is Axiom 16. The following theorem demonstrates that our model satisfies this axiom.

Theorem 6-3. For all x in D there exists a chain $\{x_k\}$ of finite elements of D such that

$$x = \bigsqcup_{k=0}^{\infty} x_k.$$

Proof: We will construct the $\{x_k\}$ for an arbitrary $x = \langle u_n \rangle_{n=1}^{\infty}$. Let each $x_k = \langle u_{k,n} \rangle_{n=1}^{\infty}$, where the $u_{k,n}$ are defined as follows. If $n < 2^{k+1}$ then $u_{k,n} = u_n$. The remaining $u_{k,n}$ are given by

$$u_{k,n+\rho(n)} = u_{k,n+2\rho(n)} = \begin{cases} \perp & \text{if } u_{k,n} = \perp \text{ or } u_{k,n} = \gamma, \\ \mathcal{E} & \text{if } u_{k,n} \neq \perp \text{ and } u_{k,n} \neq \gamma, \end{cases}$$

where $2^k \leq n$. It is easy to verify that each x_k is a object in D. Also, each x_k is finite since there is only a finite number of different n such that $u_{k,n} = \gamma$ for a given k.

To show that the x_k form a chain, we must show that $u_{k,n} \sqsubseteq^M u_{k+1,n}$ for each non-negative k and positive n. This is done by induction on n. The base case of $n < 2^{k+1}$ clearly holds since both $u_{k,n}$ and $u_{k+1,n}$ are equal to u_n. If $n \geq 2^{k+1}$ then either $u_{k,n} = \perp$ or $u_{k,n} = \mathcal{E}$. If $u_{k,n} = \perp$ then clearly $u_{k,n} \sqsubseteq^M u_{k+1,n}$. For $u_{k,n} = \mathcal{E}$, notice that there exists an $n' < n$ such that either $n' + \rho(n') = n$ or $n' + 2\rho(n') = n$. It must be the case that $u_{k,n'} \neq \perp$ and $u_{k,n'} \neq \gamma$. By the induction hypothesis, this implies that $u_{k+1,n'} \neq \perp$ and $u_{k+1,n'} \neq \gamma$. Therefore, $u_{k+1,n} = \mathcal{E}$ and so $u_{k,n} = u_{k+1,n}$.

To see that x is the least upper bound of the x_k, notice that for all n there exists a k_0 such that $u_{k,n} = u_n$ for all $k \geq k_0$. Thus,

$$u_n = \bigsqcup_{k=0}^{\infty} u_{k,n}$$

for all n. Therefore, the desired result follows from Theorem 6-2. □

We have shown that there is at least one model of $\mathcal{L}$. In the remainder of this section we show that, up to isomorphism, there is exactly one model of $\mathcal{L}$. Let $(D, \mathcal{D})$ and $(D', \mathcal{D}')$ each be an ordered pair of a domain and a semantic function that satisfy those axioms of $\mathcal{L}$ listed at the beginning of this section. As discussed earlier, each such pair can be extended in exactly one way to form a model of $\mathcal{L}$. The resulting models are isomorphic if and only if $(D, \mathcal{D})$ and $(D', \mathcal{D}')$ are isomorphic. Therefore, we can show that any two models of $\mathcal{L}$ are isomorphic by showing that $(D, \mathcal{D})$ and $(D', \mathcal{D}')$ are isomorphic.

Let h_0 be a function from finite objects in D to D' as follows. If a is an atom, then $h_0(a) = a$. Also, $h_0(\perp^D) = \perp^{D'}$, and similarly for $\Diamond^D$ and $\mathcal{E}^D$. We define h_0 for finite constructed objects $x :^{\mathcal{D}} y$ by induction on their rank,

$$h_0(x :^{\mathcal{D}} y) = h_0(x) :^{\mathcal{D}'} h_0(y).$$

The function h_0 has several important properties. It is easy to verify that x is constructed if and only if $h_0(x)$ is constructed. If x is finite, then so is $h_0(x)$. Also, for any finite x' in D', there exists a finite x in D such that $h_0(x) = x'$. It can also be shown, by induction on the rank of finite x and y in D, that $x \sqsubseteq^D y$ if and only if $h_0(x) \sqsubseteq^{D'} h_0(y)$, so h_0 is monotonic. It follows that $x = y$ if and only if $h_0(x) = h_0(y)$.

Let $\{x_k\}$ and $\{y_k\}$ be chains of finite objects in D with least upper bounds of x and y, respectively. Then $\{h_0(x_k)\}$ and $\{h_0(y_k)\}$ are chains of finite objects in

D'. Let x' and y' be the least upper bounds of these chains. Then,

$$\begin{aligned}
x \sqsubseteq^D y &\iff \left(\bigsqcup_{j=0}^{\infty} x_j\right) \sqsubseteq^D \left(\bigsqcup_{k=0}^{\infty} y_k\right) && \text{given} \\
&\iff \forall j \exists k [x_j \sqsubseteq^D y_k] && \text{Theorem 5-3} \\
&\iff \forall j \exists k [h_0(x_j) \sqsubseteq^{D'} h_0(y_k)] && \text{prop. of } h_0 \\
&\iff \left(\bigsqcup_{j=0}^{\infty} h_0(x_j)\right) \sqsubseteq^{D'} \left(\bigsqcup_{k=0}^{\infty} h_0(y_k)\right) && \text{Theorem 5-3} \\
&\iff x' \sqsubseteq^{D'} y' && \text{given.}
\end{aligned}$$

It follows that $x = y$ if and only if $x' = y'$. Therefore, it makes sense to define a function h from D to D' such that

$$h(\bigsqcup_{k=0}^{\infty} x_k) = \bigsqcup_{k=0}^{\infty} h_0(x_k)$$

for any chain $\{x_k\}$ of finite objects in D. By Axiom 16, h is defined on all elements of D. Since $x \sqsubseteq^D y$ if and only if $x' \sqsubseteq^{D'} y'$ above, it follows that h is monotonic and one-to-one. It is also easy to verify that $h(x) = h_0(x)$ for all finite x in D.

Theorem 6-4. The function h is an isomorphism from $(D, \mathcal{D})$ to $(D', \mathcal{D}')$.

Corollary. Any two models of $\mathcal{L}$ are isomorphic.

Proof: The corollary follows from the discussion in the first paragraph following Theorem 6-3. The first step in showing that h is an isomorphism is showing that it is a continuous bijection. Proving that h is a bijection requires showing that for all x' in D' there exists an x in D such that $h(x) = x'$, since we have already shown that h is one-to-one. Let x' be an object in D'. There exists a chain $\{x'_k\}$ of finite objects in D' with x' as its least upper bound. It is a property of h_0 that there exists a chain $\{x_k\}$ of finite objects in D such that $h_0(x_k) = x'_k$ for each k. Let x be the least upper bound of $\{x_k\}$. Then, $h(x) = x'$. Therefore, h is a bijection.

To prove that h is continuous, let $\{y_k\}$ be chain of objects in D with a least upper bound of y. Then $\{h(y_k)\}$ is a chain of objects in D'. Let y' be the least upper bound of this chain. We must show that $h(y) = y'$. Since h is monotonic, we know that $h(y_k) \sqsubseteq h(y)$ for each k. Therefore, $y' \sqsubseteq h(y)$. To show that $h(y) \sqsubseteq y'$, let $\{x_k\}$ be chain of finite objects in D that also has y as its least upper bound. Then $\{h(x_k)\}$ is a chain of objects in D'. Let x' be the least upper bound of this chain. Since each of the x_k is finite, we know that $h(x_k) = h_0(x_k)$. Therefore, $h(y) = x'$, so we need only show that $x' \sqsubseteq y'$. This follows from

$$\begin{aligned}
\left(\bigsqcup_{j=0}^{\infty} x_j\right) = \left(\bigsqcup_{k=0}^{\infty} y_k\right) &\Rightarrow \forall j \exists k [x_j \sqsubseteq^D y_k] && \text{Theorem 5-3} \\
&\Rightarrow \forall j \exists k [h(x_j) \sqsubseteq^{D'} h(y_k)] && \text{mon. of } h \\
&\Rightarrow \left(\bigsqcup_{j=0}^{\infty} h(x_j)\right) \sqsubseteq^{D'} \left(\bigsqcup_{k-0}^{\infty} h(y_k)\right) && \text{Theorem 5-3} \\
&\Rightarrow x' \sqsubseteq^{D'} y' && \text{given.}
\end{aligned}$$

Therefore, h is continuous.

Next we must show that h preserves the classes specified within domains: $\perp$, $\mathcal{E}$, $\Diamond$, atoms, and constructed objects. This property is clear for h applied to finite objects x, since in that case $h(x) = h_0(x)$. Thus, we need only show that if x is an infinite object, then $h(x)$ is a constructed object. Any chain $\{x_j\}$ of finite objects that has x as its least upper bound must contain some object, call it x_k, that is a constructed object. Thus, $h(x)$ is the least upper bound of a chain that contains the constructed object $h_0(x_k)$. Therefore, $h(x)$ is a constructed object.

Next we must verify that $h(x :^{\mathcal{D}} y) = h(x) :^{\mathcal{D}'} h(y)$ for all x and y in D. Let $\{x_k\}$ and $\{y_k\}$ be chains of finite objects in D with least upper bounds of x and y, repectively. Then,

$$\begin{aligned}
h(x :^{\mathcal{D}} y) &= h((\textstyle\bigsqcup_{k=0}^{\infty} x_k) :^{\mathcal{D}} (\bigsqcup_{k=0}^{\infty} y_k)) && \text{given} \\
&= h(\textstyle\bigsqcup_{k=0}^{\infty}(x_k :^{\mathcal{D}} y_k)) && \text{cont. of } :^{\mathcal{D}} \\
&= \textstyle\bigsqcup_{k=0}^{\infty} h_0(x_k :^{\mathcal{D}} y_k) && \text{def. of } h \\
&= \textstyle\bigsqcup_{k=0}^{\infty}(h_0(x_k) :^{\mathcal{D}'} h_0(y_k)) && \text{def. of } h_0 \\
&= (\textstyle\bigsqcup_{k=0}^{\infty} h_0(x_k)) :^{\mathcal{D}'} (\bigsqcup_{k=0}^{\infty} h_0(y_k)) && \text{cont. of } :^{\mathcal{D}'} \\
&= h(\textstyle\bigsqcup_{k=0}^{\infty} x_k) :^{\mathcal{D}'} h(\bigsqcup_{k=0}^{\infty} y_k) && \text{def. of } h \\
&= h(x) :^{\mathcal{D}'} h(y) && \text{given.}
\end{aligned}$$

Finally, we must verify that $h(hd^{\mathcal{D}}(x)) = hd^{\mathcal{D}'}(h(x))$ and that $h(tl^{\mathcal{D}}(x)) = tl^{\mathcal{D}'}(h(x))$ for all x in D. First we consider the case for hd. The desired property is easy to verify for unconstructed objects. For constructed objects,

$$\begin{aligned}
h(hd^{\mathcal{D}}(x :^{\mathcal{D}} y)) &= h(x) && \text{Axiom 7} \\
&= hd^{\mathcal{D}'}(h(x) :^{\mathcal{D}'} h(y)) && \text{Axiom 7} \\
&= hd^{\mathcal{D}'}(h(x :^{\mathcal{D}} y)) && \text{prop. of } h.
\end{aligned}$$

The case for tl is completely analogous. □

This completes the proof that there is exactly on model of $\mathcal{L}$, up to isomorphism. Using the notation of [1], the domain of this model can be described (up to isomorphism) by the recursive domain equation

$$D = (Atoms \cup \{\mathcal{E}, \Diamond, \perp\}) \oplus (D \times_{\perp} D).$$

Thus, the semantics given by $\mathcal{L}$ correspond to the implementation-oriented semantics given in [4].

The axioms of $\mathcal{S}$ also determine exactly one model, up to isomorphism. The proof of this is straightforward and will not be given here.

7. An Interpreter with Non-Strict Semantics

In this section we present an interpreter for the functional programming language L that has its semantics given by the axioms of $\mathcal{L}$. We also give a detailed sketch of a proof of the correctness of this interpreter.

The interpreter is written in a procedural languge based on Dijkstra's guarded commands [3]. We extend this procedural language with the addition of function calls. The proofs in this section are not based on a formal semantics of this procedural language, but instead appeal to the reader's intuition concerning the meaning of procedural programs. However, the proofs are given in enough detail that it should be straightforward to formalize them to any desired degree. See [7] for proof techniques that can be applied to recursive procedures and functions in this language. Also, see [8] for techniques that do not require functions to be rewritten as procedures.

Any issues related to the run-time environment provided by the interpreter are beyond the scope of this paper. We simply implement a function *Eval* such that when $Eval(E)$ terminates it returns a canonical (see Section 3) term equivalent to E. The function *Eval* makes use of the function *EvalCons*. When $EvalCons(E)$ terminates, it returns a term E' that is either a base term or a term that has *cons* as its root. It does no further processing on the proper subterms of E'.

The term that the interpreter should return depends on the declarations in the program being evaluated. Thus, *Eval* and *EvalCons* implicitly depend on the declarations of the program. Throughout this section we assume that *Eval* is being called on a program term that is meaningful in the value environment of some program P that has n declarations of the form

$$f(v_1, \ldots, v_{r_k}) \Leftarrow R_k[v_1, \ldots, v_{r_k}],$$

where k is between 1 and n. Whenever we use the symbols $\equiv$ and $\sqsubseteq$ as abbreviations for $\equiv_S$ and $\sqsubseteq_S$ in this section, it is understood that S is the singleton set consisting of η^P, the value environment of P. We write $E = E'$ to assert that E and E' are identical terms.

In order to represent terms in our procedural language, we add three new data types. The first two of these data types are scalar types that would be declared by

$$\begin{aligned} TermType &= [ERROR, NIL, BOOL, INT] \\ RootType &= [CONS, HD, TL, FUNCVAR, PLUS, MINUS, TIMES, \\ &\qquad MOD, EQUALP, LTEQP, ATOMP, NILP] \end{aligned}$$

in PASCAL. The scalars of type *TermType* correspond to the different kinds of base terms in L. The scalars of type *RootType* correspond to the different roots a non-base term in L can have. The third data type is called *term*. Rather than give the details of the structure of the type *term*, we assume the existence of predefined functions that provide abstract operations on this data type. Informal semantics of each of these predefined functions is given in Table 7-1. This table shows the result of a call to each function, given that the arguments to the functions satisfy

Function	Restrictions	Result
$IsBase(E : term) : Boolean$	none	*true* if E is a base term, *false* otherwise
$Root(E : term) : RootType$	non-base term	scalar corresponding to the root of E
$TypeOf(E : term) : TermType$	base term	scalar corresponding to the type of E
$BoolValue(E : term) : Boolean$	bool. base term	bool. value equiv. to E
$IntValue(E : term) : Integer$	int. base term	int. value equiv. to E
$Subterm(n : Integer; E : term) : term$	$E = g(E_1, \ldots, E_r)$ $1 \leq n \leq r$	E_n
$ConsTerm(E_1, E_2 : term) : term$	none	$E_1 : E_2$
$ErrorTerm : term$	none	$\mathcal{E}$
$ApplyCopyRule(E : term) : term$	$E = f_k(E_1, \ldots, E_{r_k})$	$R_k[E_1, \ldots, E_{r_k}]$
$BoolTerm(b : Boolean) : term$	none	base term equiv. to b
$IntTerm(n : Integer) : term$	none	base term equiv. to n

Table 7-1. Predefined functions used in *Eval* and *EvalCons*.

the restrictions in the second column. Notice that the function *ApplyCopyRule* depends implicitly on the declarations of P.

Clearly any call to *Eval* or to *EvalCons* that terminates makes only a finite number of recursive function calls. In particular, only a finite number of recursive function calls to *Eval* or *EvalCons* are made. Thus, we can use induction on the number of recursive calls to *Eval* or *EvalCons* in order to prove properties of terminating computations. The following two theorems prove important properties *Eval* and *EvalCons* using this method.

Theorem 7-1. If E is a variable-free term and $EvalCons(E)$ terminates, then

1) $EvalCons(E) \equiv E$, and
2) $EvalCons(E)$ is a variable-free term that either is a base term or has *cons* as its root.

Proof: Since $EvalCons(E)$ is assumed to terminate, we can prove the theorem by induction on the number of recursive calls to *EvalCons*. In the base case there are no recursive calls to *EvalCons*, which implies that E is a base term or has *cons* as its root. It follows that $EvalCons(E) = E$, so the theorem holds in the base case.

For the induction step, assume the number of recursive calls to *EvalCons* is greater than zero. This implies that E is a non-base term with a root other than *cons*. The proof of the induction step is by case analysis on the possible roots of E.

```
1)  Function Eval(value E : term) : term;
2)      var E' : term;
3)      begin
4)          E' := EvalCons(E);
5)          if IsBase(E') ⟶ result := E'
6)          [] ¬IsBase(E') ⟶ {Root(E') = CONS}
7)              result := ConsTerm(Eval(Subterm(1, E')), Eval(Subterm(2, E')))
8)          fi
9)      end
```

```
1)  Function EvalCons(value E : term) : term;
2)      var n : Integer;  b : Boolean;  E', E'' : term;
3)      begin
4)          if IsBase(E) ⟶ result := E
5)          [] ¬IsBase(E) ⟶
6)              if Root(E) = CONS ⟶ result := E
7)              [] Root(E) in [HD, TL] ⟶
8)                  E' := EvalCons(Subterm(1, E));
9)                  if IsBase(E') ⟶ result := ErrorTerm
10)                 [] ¬IsBase(E') ⟶ {Root(E') = CONS}
11)                     if Root(E) = HD ⟶ E'' := Subterm(1, E')
12)                     [] Root(E) = TL ⟶ E'' := Subterm(2, E')
13)                     fi;
14)                     result := EvalCons(E'')
15)                 fi
16)             [] Root(E) = IF ⟶
17)                 E' := EvalCons(Subterm(1, E));
18)                 if IsBase(E') ⟶
19)                     if TypeOf(E') = BOOL ⟶
20)                         if BoolValue(E') ⟶ E'' := Subterm(2, E)
21)                         [] ¬BoolValue(E') ⟶ E'' := Subterm(3, E)
22)                         fi;
23)                         result := EvalCons(E'')
24)                     [] TypeOf(E') ≠ BOOL ⟶ result := ErrorTerm
25)                     fi
26)                 [] ¬IsBase(E') ⟶ result := ErrorTerm
27)                 fi
28)             [] Root(E) = FUNCVAR ⟶
29)                 result := EvalCons(ApplyCopyRule(E))
```

```
30)        [] Root(E) in [PLUS, MINUS, TIMES, MOD, EQUALP, LTEQP] ⟶
31)            E' := EvalCons(Subterm(1, E));
32)            if IsBase(E') ⟶
33)                if TypeOf(E') = INT ⟶
34)                    E'' := EvalCons(Subterm(2, E));
35)                    if IsBase(E'') ⟶
36)                        if TypeOf(E'') = INT ⟶
37)                            if Root(E) in [PLUS, MINUS, TIMES] ⟶
38)                                if Root(E) = PLUS ⟶
39)                                    n := IntValue(E') + IntValue(E'')
40)                                [] Root(E) = MINUS ⟶
41)                                    n := IntValue(E') − IntValue(E'')
42)                                [] Root(E) = TIMES ⟶
43)                                    n := IntValue(E') × IntValue(E'')
44)                                fi;
45)                                result := IntTerm(n)
46)                            [] Root(E) = MOD ⟶
47)                                n := IntValue(E'');
48)                                if n ≤ 0 ⟶ result := ErrorTerm
49)                                [] n > 0 ⟶
50)                                    n := IntValue(E') mod n;
51)                                    result := IntTerm(n)
52)                                fi
53)                            [] Root(E) in [EQUALP, LTEQP] ⟶
54)                                if Root(E) = EQUALP ⟶
55)                                    b := (IntValue(E') = IntValue(E''))
56)                                [] Root(E) = LTEQP ⟶
57)                                    b := (IntValue(E') ≤ IntValue(E''))
58)                                fi;
59)                                result := BoolTerm(b)
60)                            fi
61)                        [] TypeOf(E'') ≠ INT ⟶ result := ErrorTerm
62)                        fi
63)                    [] ¬IsBase(E'') ⟶ result := ErrorTerm
64)                    fi
65)                [] TypeOf(E') ≠ INT ⟶ result := ErrorTerm
66)                fi
67)            [] ¬IsBase(E') ⟶ result := ErrorTerm
68)            fi
```

```
69)              [] Root(E) in [ATOMP, NILP] ⟶
70)                 E' := EvalCons(Subterm(1, E));
71)                 if IsBase(E') ⟶
72)                    if TypeOf(E') = ERROR     › result := ErrorTerm
73)                    [] TypeOf(E') ≠ ERROR ⟶
74)                       if Root(E) = ATOMP ⟶
75)                          b := TypeOf(E') in [INT, BOOL]
76)                       [] Root(E) = NILP ⟶
77)                          b := TypeOf(E') = NIL
78)                       fi;
79)                       result := BoolTerm(b)
80)                    fi
81)                 [] ¬IsBase(E') ⟶ result := BoolTerm(false)
82)                 fi
83)              fi
84)           fi
85)        end
```

If the function symbol *hd* is the root of E, then by the induction hypothesis E' is set to a term that is either a base term or has *cons* as its root. The induction hypothesis also implies that $E \equiv hd(E')$. If E' is a base term, then $E \equiv \mathcal{E}$ and $EvalCons(E) = \mathcal{E}$, so the theorem holds. Otherwise the induction hypothesis gives the desired result since $hd(E') \equiv Subterm(1, E')$ when *cons* is the root of E'. If *tl* is the root of E, then the proof is completely analogous.

If E is of the form **if** E_1 **then** E_2 **else** E_3, then E' is set to a term that is equivalent to E_1. If E' is not a boolean base term, then $EvalCons(E) = \mathcal{E}$, which satisfies the theorem. If $E' = true$ then $E \equiv Subterm(2, E)$. If $E' = false$ then $E \equiv Subterm(3, E)$. So it follows from the induction hypothesis that the theorem holds in these cases as well.

If the root of E is a function variable, then the desired result follows easily from the induction hypothesis, since $ApplyCopyRule(E) \equiv E$.

The remaining cases are left as an exercise for the reader. □

Theorem 7-2. If E is a variable-free term and $Eval(E)$ terminates, then

1) $Eval(E) \equiv E$, and
2) $Eval(E)$ is a canonical term.

Proof: Since $EvalCons(E)$ is assumed to terminate, we can prove the theorem by induction on the number of recursive calls to *Eval*. In the base case there are no recursive calls to *Eval*, which implies that E' is set to a base term. It follows from Theorem 7-1 that $E' \equiv E$, so the theorem holds in the base case.

For the induction step, assume the number of recursive calls to *Eval* is greater than zero. This implies that E' is set to a term that has *cons* as its root. Therefore, $Eval(Subterm(1, E'))$ is equivalent to $hd(E)$ and is a canonical term. The analogous

statement is also true of $Eval(Subterm(2, E'))$. Therefore, $Eval(E)$ is equivalent to E and is a canonical term. □

The next several theorems are used in the proof of Theorem 7-7, which states that $EvalCons(E)$ terminates if and only in $E \not\equiv \perp$. We start by proving that $EvalCons(E)$ terminates for a smaller class of E.

Theorem 7-3. If E is a variable-free term such that $\mathcal{D}[\![E]\!]\eta_0^P \neq \perp$, then

1) $EvalCons(E)$ terminates, and
2) if $EvalCons(E)$ is not a base term, then it is a subterm of E.

Corollary. If E is a variable-free term such that $E \not\equiv \perp$, then there exists a non-negative integer n such that $EvalCons(\Phi^n(E))$ terminates.

Proof: First we show that the corollary follows from the theorem. Let E be a variable-free term not equivalent to $\perp$. It follows from Theorem 3-15 that

$$\mathcal{D}[\![\Phi^n[\![E]\!]]\!]\eta_0^P = \mathcal{D}[\![E]\!]\eta_n^P$$

for all non-negative integers n. Therefore, there exists a non-negative n such that $\mathcal{D}[\![\Phi^n[\![E]\!]]\!]\eta_0^P \neq \perp$.

To prove the theorem, let E be a variable-free term such that $\mathcal{D}[\![E]\!]\eta_0^P \neq \perp$. The proof is by induction on the structure of E. The base case is trivial since $EvalCons(E)$ obviously terminates and is equal to E when E is a base term.

The induction step is proven by case analysis on each of the possible roots of E. If *cons* is the root of E, then again $EvalCons(E)$ obviously terminates and is equal to E.

Let E be of the form $hd(E_1)$. Clearly $\mathcal{D}[\![E_1]\!]\eta_0^P \neq \perp$, so, by the induction hypothesis the statement $E' := EvalCons(Subterm(1, E))$ terminates and sets E' to a term that is equivalent to E_1. If E' is a base term, then clearly $EvalCons(E)$ terminates and is a base term. Otherwise, the root of E' is *cons*, so by the induction hypothesis, E' is a subterm of E. Thus, E'' is set to a term that is equivalent to E, and is also a subterm of E. Therefore, the desired result follows from the induction hypothesis since $EvalCons(E)$ is computed by computing $EvalCons(E'')$. The proof is completely analogous when *tl* is the root of E.

Let E be of the form **if** E_1 **then** E_2 **else** E_3. As in the *hd* case, E' is set to a term that is equivalent to E_1. If E' is not a boolean base term, then clearly $EvalCons(E)$ terminates and is a base term. Otherwise, E'' is set to a term that is equivalent to E, and is also a subterm of E. Therefore, the desired result follows from the induction hypothesis just as in the *hd* case.

If the root of E is a function variable, then clearly $\mathcal{D}[\![E]\!]\eta_0^P = \perp$, which contradicts the assumptions of the theorem.

The remaining cases are left as an exercise for the reader. □

The corollary of the previous theorem is an important result for proving that $EvalCons(E)$ terminates if and only if $E \not\equiv \perp$. This result follows from the corollary

if we show (as we do in Theorem 7-6) that $EvalCons(E)$ terminates if and only if $EvalCons(\Phi[\![E]\!])$ terminates. For this proof we define a total function Ψ. The domain of Ψ is the set of extraction functions. The codomain of Ψ is the set of partial functions from variable-free terms to variable-free terms. Applied to the identity extraction function Ψ returns the identity function from variable-free terms to variable-free terms. If e is an extraction function and E is a variable-free term such that $EvalCons(\Psi(e)(E))$ terminates and returns a non-base term (which, therefore, has *cons* as its root), then

$$\Psi(hd \circ e)(E) = Subterm(1, EvalCons(\Psi(e)(E)))$$

and

$$\Psi(tl \circ e)(E) = Subterm(2, EvalCons(\Psi(e)(E))),$$

otherwise $\Psi(hd \circ e)(E)$ and $\Psi(tl \circ e)(E)$ are undefined. Thus, if e is an extraction function over the domain D, then $\Psi(e)$ is the corresponding "extraction" function over terms.

We say $E \sim E'$ when $E \equiv E'$ and for all e such that $\Psi(e)(E)$ and $\Psi(e)(E')$ are defined, $EvalCons(\Psi(e)(E))$ terminates if and only if $EvalCons(\Psi(e)(E'))$ terminates. This relation is clearly reflexive and symmetric. It also has the property that if $E \sim E'$, then for all extraction functions e, $\Psi(e)(E)$ is defined if and only if $\Psi(e)(E')$ is defined. It follows then that $\sim$ is transitive, and is, therefore, an equivalence relation.

We call $R[v_1, \ldots, v_r]$ a *unifying term for E and E'* when there exist terms E_i, E_i' for $1 \leq i \leq r$ such that $E = R[E_1, \ldots, E_r]$, $E' = R[E_1', \ldots, E_r']$, and $E_i \sim E_i'$ for $1 \leq i \leq r$. We say $E \approx E'$ when there exists a unifying term for E and E'. Notice that $E \approx E'$ implies $E \equiv E'$. Clearly $\approx$ is reflexive and symmetric. It will become clear that it is also transitive only when we prove (in Theorem 7-5) that $E \sim E'$ if and only if $E \approx E'$.

There are several facts about $\sim$ and $\approx$ that we will use. If $E \sim E'$ then $E \approx E'$. If $E \sim E'$ and $EvalCons(E)$ terminates, then $EvalCons(E')$ terminates and $EvalCons(E) \sim EvalCons(E')$. If E and E' have a base unifying term, then $E \sim E'$. If E and E' have a non-base unifying term, then $Subterm(n, E) \approx Subterm(n, E')$ for any n for which both $Subterm(n, E)$ and $Subterm(n, E')$ are defined. It is more difficult to show that if $E \approx E'$ and E and E' have *cons* as their root, then $Subterm(1, E) \approx Subterm(1, E')$ and $Subterm(2, E) \approx Subterm(2, E')$. This is clearly true when E and E' have a non-base unifying term. Otherwise, $E \sim E'$ which implies $Subterm(1, E) \sim Subterm(1, E')$ and $Subterm(2, E) \sim Subterm(2, E')$. Another important property of $\approx$ is stated in the following theorem.

Theorem 7-4. If E_1 and E_2 are variable-free terms such that $E_1 \approx E_2$, then

1) $EvalCons(E_1)$ terminates if an only if $EvalCons(E_2)$ terminates, and
2) if both $EvalCons(E_1)$ and $EvalCons(E_2)$ terminate, then $EvalCons(E_1) \approx EvalCons(E_2)$.

Proof: We consider three (overlapping) cases: either $EvalCons(E_1)$ terminates, or $EvalCons(E_2)$ terminates, or both $EvalCons(E_1)$ and $EvalCons(E_2)$ fail to terminate. The theorem holds trivially in the third case. By symmetry, we need only

consider the first of the remaining two cases. The proof of this case is by induction on the number of recursive calls to *EvalCons* made during the computation of $EvalCons(E_1)$. Since $E_1 \approx E_2$, there exists a unifying term $R[v_1, \ldots, v_r]$.

For the base case of the induction, the number of recursive calls to *EvalCons* is zero. In this case, E_1 is either a base term, or has *cons* as its root. Therefore, $R[v_1, \ldots, v_r]$ is either a base term, or has *cons* as its root. If $R[v_1, \ldots, v_r]$ is a base term, then $E_1 \sim E_2$. If $R[v_1, \ldots, v_r]$ has *cons* as its root then both $EvalCons(E_1)$ and $EvalCons(E_2)$ terminate leaving $EvalCons(E_1) = E_1$ and $EvalCons(E_2) = E_2$.

For the induction step, the number of recursive calls to *EvalCons* is greater than zero. If $R[v_1, \ldots, v_r]$ is a base term, then again $E_1 \sim E_2$. Otherwise, we will consider cases depending on the root of $R[v_1, \ldots, v_r]$. Notice that we need not consider the case of the root being *cons* since we assumed that the number of recursive calls to *EvalCons* is greater than zero.

If the root is *hd*, then $Subterm(1, E_1) \approx Subterm(1, E_2)$. It follows from the induction hypothesis that $EvalCons(Subterm(1, E_2))$ terminates, and that

$$EvalCons(Subterm(1, E_1)) \approx EvalCons(Subterm(1, E_2)).$$

Let $E_1' = EvalCons(Subterm(1, E_1))$ and let $E_2' = EvalCons(Subterm(1, E_2))$. If one these terms is a base term, then they both are. In that case $EvalCons(E_1)$ and $EvalCons(E_2)$ both terminate, and are equal to $\mathcal{E}$. Otherwise, both terms have *cons* as their root, so $Subterm(1, E_1') \approx Subterm(1, E_2')$ and the theorem follows from the induction hypothesis. The proof is completely analogous when *tl* is the root.

If the root of $R[v_1, \ldots, v_r]$ is **if**, then $Subterm(1, E_1) \approx Subterm(1, E_2)$. It follows from the induction hypothesis that $EvalCons(Subterm(1, E_2))$ terminates. If $EvalCons(E_1)$ is a boolean base term, then so is $EvalCons(E_2)$. Both terms will have the same boolean value, and for either boolean value the theorem follows from the induction hypothesis. Otherwise $EvalCons(E_1)$ and $EvalCons(E_2)$ both terminate, and are equal to $\mathcal{E}$.

If the root of $R[v_1, \ldots, v_r]$ is a function variable, then $ApplyCopyRule(E_1) \approx ApplyCopyRule(E_2)$. Therefore, the theorem follows from the induction hypothesis.

The remaining cases are left as an exercise for the reader. □

Theorem 7-5. If E and E' are variable-free terms, then $E \approx E'$ if and only if $E \sim E'$.

Proof: The reverse implication has already been discussed. To prove the forward implication, assume that E and E' are variable-free terms such that $E \approx E'$. We must show that for all extraction functions e such that $\Psi(e)(E)$ and $\Psi(e)(E')$ are defined, $EvalCons(\Psi(e)(E))$ terminates if and only if $EvalCons(\Psi(e)(E'))$ terminates. By Theorem 7-4 it is sufficient to show that $\Psi(e)(E) \approx \Psi(e)(E')$ for all such e. The proof is by induction on the rank of e. If e is the identity function, then $\Psi(e)$ is also the identity function, so $\Psi(e)(E) \approx \Psi(e)(E')$.

If $\Psi(hd \circ e)(E)$ and $\Psi(hd \circ e)(E')$ are defined, then so are $\Psi(e)(E)$ and $\Psi(e)(E')$. Also, $EvalCons(\Psi(e)(E))$ and $EvalCons(\Psi(e)(E'))$ terminate, and both of these

terms have *cons* as their root. By the induction hypothesis, $\Psi(e)(E) \approx \Psi(e)(E')$. Thus,

$$EvalCons(\Psi(e)(E)) \approx EvalCons(\Psi(e)(E'))$$

by Theorem 7-4. Therefore, $\Psi(hd \circ e)(E) \approx \Psi(hd \circ e)(E')$.

The proof for $tl \circ e$ is completely analogous. □

We introduced the relations $\sim$ and $\approx$ strictly for the purpose of proving the following theorem.

Theorem 7-6. If E is a variable-free term, then $\Phi[\![E]\!] \sim E$.

Corollary. If E is a variable-free term, then $EvalCons(E)$ terminates if and only if $EvalCons(\Phi[\![E]\!])$ terminates.

Proof: The corollary follows immediately from the definition of $\sim$. The proof of the theorem is by induction on the structure of E. If E is a base term, then $\Phi[\![E]\!] = E$. If E has a function symbol as its root, it follows from the induction hypothesis that $\Phi[\![E]\!] \approx E$, so $\Phi[\![E]\!] \sim E$ by Theorem 7-5.

Otherwise E is of the form $f_k(E_1, \ldots, E_{r_k})$. By the induction hypothesis,

$$f_k(E_1, \ldots, E_{r_k}) \approx f_k(\Phi[\![E_1]\!], \ldots, \Phi[\![E_{r_k}]\!]).$$

The execution of $EvalCons(f_k(\Phi[\![E_1]\!], \ldots, \Phi[\![E_{r_k}]\!]))$ is essentially identical to the execution of $EvalCons(\Phi[\![E]\!])$ since

$$ApplyCopyRule(f_k(\Phi[\![E_1]\!], \ldots, \Phi[\![E_{r_k}]\!])) = \Phi[\![E]\!].$$

Thus, $f_k(\Phi[\![E_1]\!], \ldots, \Phi[\![E_{r_k}]\!]) \sim \Phi[\![E]\!]$. Therefore, $\Phi[\![E]\!] \sim E$ by Theorem 7-5 and the transitivity of $\sim$. □

Theorem 7-7. If E is a variable-free term, then $EvalCons(E)$ terminates if and only if $E \not\equiv \perp$.

Proof: The forward implication follows from Theorem 7-1 since there is no base term that is equivalent to $\perp$. The reverse implication follows from the corollaries of Theorems 7-3 and 7-6. □

Now that we have shown under what conditions *EvalCons* terminates, it is straightforward to show under what conditions *Eval* terminates.

Theorem 7-8. If E is a variable-free term, then $Eval(E)$ terminates if and only if E is equivalent to some canonical term.

Proof: The forward implication follows from Theorem 7-2. The reverse implication is proven by induction on the structure of canonical terms. □

We axiomatized *cons* in $\mathcal{L}$ so that $x : y$ is (equivalent to) a constructed object for all x and y. This requires some justification. When choosing the axioms for

cons there are sixteen special cases to consider. These cases are listed below, where a is an atom and z is either $\Diamond$ or a constructed object.

1) $\bot:\bot$	5) $\mathcal{E}:\bot$	9) $a:\bot$	13) $z:\bot$
2) $\bot:\mathcal{E}$	6) $\mathcal{E}:\mathcal{E}$	10) $a:\mathcal{E}$	14) $z:\mathcal{E}$
3) $\bot:a$	7) $\mathcal{E}:a$	11) $a:a$	15) $z:a$
4) $\bot:z$	8) $\mathcal{E}:z$	12) $a:z$	16) $z:z$

When discussing these cases we will often make use of the fact that if $x \sqsubseteq y$ and x is a constructed object, then y is a constructed object.

Any conventional semantics for *cons* requires that $a:z$ and $z:z$ be constructed objects. This implies that $\bot:z$, $a:\bot$, and $z:\bot$ are either equivalent to $\bot$ or are constructed objects. In the strict semantics all of these are equivalent to $\bot$. For the non-strict semantics we chose to axiomatize *cons* so that all of these are constructed objects. This is the standard choice for non-strict semantics. It follows from the monotonicity of *cons* and the fact noted above that cases 4, and 8 thru 16, are all constructed objects.

The remaining cases depend on whether we require that the semantics be implementable by a *serial* interpreter, or whether we allow the interpreter to be *parallel*. A serial interpreter evaluates a term E by evaluating some subterm, and then depending on the result it may evaluate more subterms. If the evaluation of the chosen subterm of E does not terminate, then the evaluation of E does not terminate. A parallel interpreter may do simultaneous evaluations of two or more subterms of E, simultaneous in the sense that the evaluation of E may terminate even if one or more of the evaluations of a subterm does not terminate.

The interpreter we have described in this section is serial. We intentionally designed the semantics to make this possible. Let us consider how this restricts the semantics of *cons*, given the restrictions on cases 4, and 8 thru 16, discussed above. Let E be a non-base term with *cons* as its root. When determining whether E is a constructed object, a serial interperter may evaluate no subterms, or it may evaluate the left subterm first, or it may evaluate the right subterm first. The part of our interpreter that determines whether E is a constructed object is the procedure *EvalCons*. The call *EvalCons*(E) returns a term with *cons* as its root if and only if E is a constructed object. In doing this, *EvalCons* evaluates no subterms, this clearly requires that $x:y$ be a constructed object for all x and y.

Suppose the interpreter evaluates the left subterm of E first. Let E' be a variable-free term. The term $E':\Diamond$ is equivalent to a constructed object because of the restrictions discussed above. Thus, the interpreter must terminate and return $\Diamond$ when applied to $tl(E':\Diamond)$. Therefore, the interpreter must apply an algorithm to the left subterm that terminates on all terms. This algorithm must return the same result when applied to any two terms that are equivalent. This algorithm must also be Turing computable. The only algorithms that satisfy all these requirements are constant, they produce the same output regardless of input. This implies that this interpreter can only implement semantics for which $x:y$ is a constructed object for all x and y. Similar reasoning applies if the interpreter first evaluates the right

subterm. Therefore, our requirement that the semantics be implementable by a serial interpreter forces us to axiomatize *cons* such that $x:y$ is a constructed object for all x and y. This explains why we axiomatized *cons* in $\mathcal{L}$ as we did.

8. Example correctness proofs in $\mathcal{S}$

In this section we will show two examples of how properties of programs can be proven in $\mathcal{S}$. The statements made in this section apply only to $\mathcal{S}$. Whenever we use the symbols $\equiv$ and $\sqsubseteq$ as abbreviations for $\equiv_S$ and $\sqsubseteq_S$ in this section, it is understood that S is the set of all value environments of programs that include the declarations being considered.

The two function declarations we will be considering are

$$\begin{aligned} conc(x,y) &\Leftarrow \textbf{if } nil(x) \textbf{ then } y \\ &\qquad\quad \textbf{else } hd(x) : conc(tl(x), y) \\ rev(x) &\Leftarrow \textbf{if } nil(x) \textbf{ then } \Diamond \\ &\qquad\quad \textbf{else } conc(rev(tl(x)), hd(x) : \Diamond). \end{aligned}$$

Theorem 8-1. If z_1 and z_2 are lists of length n_1 and n_2, respectively, then $conc(z_1, z_2) \equiv z$, where z is the list of length $n_1 + n_2$ such that

$$hd(tl^k(z)) \equiv \begin{cases} hd(tl^k(z_1)) & \text{if } 0 \le k < n_1; \\ hd(tl^{k-n_1}(z_2)) & \text{if } n_1 \le k < n_1 + n_2. \end{cases}$$

Proof: Before considering the proof, notice that we know z is completely determined because of Theorem 4-4. The proof of the theorem is by induction on the length of z_1. In the base case, z_1 is a list of length zero, so $z_1 = \Diamond$. The theorem holds in this case since

$$\begin{aligned} conc(\Diamond, z_2) &\equiv \textbf{if } nil(\Diamond) \textbf{ then } z_2 && \text{def. of } conc \\ &\qquad \textbf{else } hd(\Diamond) : conc(tl(\Diamond), z_2) \\ &\equiv z_2 && \text{prop. of } nil \text{ and } \textbf{if}. \end{aligned}$$

For the induction step, consider the list $x : z_1$ where x is an atom or a list. Then,

$$\begin{aligned} conc(x:z_1, z_2) &\equiv \textbf{if } nil(x:z_1) \textbf{ then } z_2 && \text{def. of } conc \\ &\qquad \textbf{else } hd(x:z_1) : conc(tl(x:z_1), z_2) \\ &\equiv hd(x:z_1) : conc(tl(x:z_1), z_2) && \text{prop. of } nil \text{ and } \textbf{if} \\ &\equiv x : conc(z_1, z_2) && \text{prop. of } hd \text{ and } tl \\ &\equiv x : z && \text{ind. hyp.} \end{aligned}$$

Thus, $conc(x:z_1, z_2)$ is a list of length $n_1 + n_2 + 1$. So we need only show that

$$\begin{aligned} &hd(tl^k(conc(x:z_1, z_2))) \\ &\quad \equiv hd(tl^k(x:z)) && \text{previously shown} \\ &\quad \equiv \begin{cases} hd(tl^k(x:z_1)) & 0 \le k < n_1 + 1 \\ hd(tl^{k-n_1-1}(z_2)) & n_1 + 1 \le k < n_1 + n_2 + 1 \end{cases} && \text{def. of } z. \quad \square \end{aligned}$$

Theorem 8-2. If z is a list of length n, then $rev(z) \equiv z'$ where z' is the list of length n such that

$$hd(tl^k(z')) \equiv hd(tl^{n-k-1}(z))$$

for $0 \leq k < n$.

Proof: The proof is by induction on the length of z. In the base case, z is a list of length zero, so $z = \Diamond$. The theorem holds in this case since

$$\begin{aligned} rev(\Diamond) &\equiv \textbf{if } nil(\Diamond) \textbf{ then } \Diamond && \text{def. of } rev \\ &\qquad \textbf{else } conc(rev(tl(\Diamond)), hd(\Diamond) : \Diamond) \\ &\equiv \Diamond && \text{prop. of } nil \text{ and } \textbf{if} \end{aligned}$$

For the induction step, consider the list $x : z$ where x is an atom or a list. Then,

$$\begin{aligned} rev(x : z) &\equiv \textbf{if } nil(x : z) \textbf{ then } \Diamond && \text{def. of } rev \\ &\qquad \textbf{else } conc(rev(tl(x : z)), hd(x : z) : \Diamond) \\ &\equiv conc(rev(tl(x : z)), hd(x : z) : \Diamond) && \text{prop. of } nil \text{ and } \textbf{if} \\ &\equiv conc(rev(z), x : \Diamond) && \text{prop. of } hd \text{ and } tl \\ &\equiv conc(z', x : \Diamond) && \text{ind. hyp.} \end{aligned}$$

Thus, $rev(x : z)$ is a list of length $n + 1$. So we need only show that

$$\begin{aligned} hd(tl^k(rev(x : z))) &\equiv hd(tl^k(conc(z', x : \Diamond))) && \text{previously shown} \\ &\equiv \begin{cases} hd(tl^k(z')) & 0 \leq k < n \\ x & \text{k=n} \end{cases} && \text{Theorem 8-1} \\ &\equiv \begin{cases} hd(tl^{n-k-1}(z)) & 0 \leq k < n \\ x & \text{k=n} \end{cases} && \text{def. of } z' \\ &\equiv hd(tl^{n-k}(x : z)) && \text{prop. of } cons \text{ and } tl. \quad \square \end{aligned}$$

9. Example correctness proofs in $\mathcal{L}$

In this section we will show examples of how properties of programs can be proven in $\mathcal{L}$. The statements made in this section apply only to $\mathcal{L}$. As in the previous section, whenever we use the symbols $\equiv$ and $\sqsubseteq$ as abbreviations for $\equiv_S$ and $\sqsubseteq_S$ in this section, it is understood that S is the set of all value environments of programs that include the declarations being considered.

Consider the functions *conc* and *rev* as declared in the previous section. The theorems proved in that section for $\mathcal{S}$ also are true for $\mathcal{L}$, and the proofs are identical. This is an example of how our proof techniques can be applied equally well to programs with strict and non-strict semantics. The results of the previous section completely specify the behavior of *conc* and *rev* on lists of finite length. The following theorem specifies the behavior of *rev* on objects of infinite length.

Theorem 9-1. If x is an object of infinite length, then $rev(x) \equiv \bot$.

Proof: Let f be the function defined by

$$f(x) = \begin{cases} rev(x) & \text{if } x \text{ is of finite length;} \\ \bot & \text{if } x \text{ is of infinite length.} \end{cases}$$

We will show that f is a fixed point of the declaration of *rev*. Since *rev* is the least fixed point of its declaration, this implies that $rev \sqsubseteq f$. Therefore, if x is an object of infinite length, then $rev(x) \sqsubseteq \bot$, which implies that $rev(x) \equiv \bot$.

To show that f is a fixed point of the declaration of *rev*, we must show that

$$f(x) \equiv \textbf{if } nil(x) \textbf{ then } \Diamond \textbf{ else } conc(f(tl(x)), hd(x) : \Diamond), \qquad (*)$$

for all objects x. The proof has two cases depending on whether x is an object of finite length. In both cases we use the fact $tl(x)$ is of finite length if and only if x is of finite length.

If x is of finite length, then

$$\begin{aligned} f(x) &\equiv rev(x) && \text{def. of } f \\ &\equiv \textbf{if } nil(x) \textbf{ then } \Diamond && \text{def. of } rev \\ &\qquad \textbf{else } conc(rev(tl(x)), hd(x) : \Diamond) \\ &\equiv \textbf{if } nil(x) \textbf{ then } \Diamond && \text{def. of } f. \\ &\qquad \textbf{else } conc(f(tl(x)), hd(x) : \Diamond) \end{aligned}$$

If x is an object of infinite length, then

$$\begin{aligned} \text{r.h.s.}(*) &\equiv conc(f(tl(x)), hd(x) : \Diamond) && \text{prop. of } nil \text{ and } \textbf{if} \\ &\equiv conc(\bot, hd(x) : \Diamond) && \text{def. of } f \\ &\equiv \textbf{if } nil(\bot) \textbf{ then } hd(x) : \Diamond && \text{def. of } conc \\ &\qquad \textbf{else } hd(\bot) : conc(tl(\bot), hd(x) : \Diamond) \\ &\equiv \bot && \text{prop. of } nil \text{ and } \textbf{if} \\ &\equiv f(x) && \text{def. of } f. \quad \square \end{aligned}$$

Consider the function *from* declared by

$$from(n) \Leftarrow n : from(n+1).$$

Theorem 9-2. If n is an integer, then $hd(tl^k(from(n))) \equiv n + k$ for all non-negative integers k.

Proof: The proof is by induction on k. If $k = 0$, then

$$\begin{aligned} hd(tl^0(from(n))) &\equiv hd(n : from(n+1)) && \text{def. of } from \\ &\equiv n && \text{prop. of } cons \text{ and } hd. \end{aligned}$$

The induction step holds since

$$\begin{aligned} hd(tl^k(from(n))) &\equiv hd(tl^k(n : from(n+1))) && \text{def. of } from \\ &\equiv hd(tl^{k-1}(from(n+1))) && \text{prop. of } cons \text{ and } tl \\ &\equiv n+k && \text{ind. hyp. } \square \end{aligned}$$

The next function we will be considering is

$$\begin{aligned} filtcr(n,z) \Leftarrow \; & \textbf{if } (hd(z) \textbf{ mod } n) = 0 \\ & \textbf{then } filter(n, tl(z)) \\ & \textbf{else } hd(z) : filter(n, tl(z)). \end{aligned}$$

Let p be a positive integer. Let $x_0, x_1, \ldots, x_k, \ldots$ be a strictly increasing sequence of integers such that for any k_0 there exists $k \geq k_0$ so that $(x_k \bmod p) \neq 0$.

Let x'_0 be the smallest element of $\{x_j \mid j \geq 0\}$ not divisible by p. For any non-negative integer k let x'_{k+1} be the smallest element of $\{x_j \mid j \geq 0 \wedge x_j > x'_k\}$ not divisible by p. Let z and z' be the objects of infinite length such that $hd(tl^k(z)) = x_k$ and $hd(tl^k(z')) = x'_k$ for all $k \geq 0$. Let $\alpha_{-1} = -1$ and α_k be such that $x'_k = x_{\alpha_k}$ for all $k \geq 0$.

Theorem 9-3. For α_k, p, and z as defined above, if $\alpha_{k-1} < j \leq \alpha_k$ then

$$filter(p, tl^j(z)) \equiv hd(tl^{\alpha_k}(z)) : filter(p, tl^{\alpha_k+1}(z)).$$

Proof: First we show that

$$filter(p, tl^j(z)) \equiv filter(p, tl^{\alpha_k}(z))$$

by induction on the difference between α_k and j. This equivalence is trivially true for $j = \alpha_k$. If $\alpha_{k-1} < j < \alpha_k$, then

$$\begin{aligned} filter(p, tl^j(z)) \;\equiv\; & \textbf{if } (hd(tl^j(z)) \textbf{ mod } p) = 0 && \text{def. of } filter \\ & \quad \textbf{then } filter(p, tl(tl^j(z))) \\ & \quad \textbf{else } hd(tl^j(z)) : filter(p, tl(tl^j(z))) \\ \equiv\; & filter(p, tl^{j+1}(z)) && \text{prop. of } z \text{ and } \textbf{if} \\ \equiv\; & filter(p, tl^{\alpha_k}(z)) && \text{ind. hyp.} \end{aligned}$$

The theorem follows since

$$\begin{aligned} & filter(p, tl^{\alpha_k}(z)) \\ \equiv\; & \textbf{if } (hd(tl^{\alpha_k}(z)) \textbf{ mod } p) = 0 && \text{def. of } filter \\ & \quad \textbf{then } filter(p, tl(tl^{\alpha_k}(z))) \\ & \quad \textbf{else } hd(tl^{\alpha_k}(z)) : filter(p, tl(tl^{\alpha_k}(z))) \\ \equiv\; & hd(tl^{\alpha_k}(z)) : filter(p, tl^{\alpha_k+1}(z)) && \text{prop. of } z \text{ and } \textbf{if}. \; \square \end{aligned}$$

Theorem 9-4. For p, z, and z' as defined above, $filter(p, z) \equiv z'$.

Proof: We show that if $k \geq 0$ and $n \geq 0$, then

$$hd(tl^k(filter(p, tl^{\alpha_{n-1}+1}(z)))) \equiv x'_{k+n}$$

by induction on k. The theorem follows from this since $\alpha_{-1} = -1$. The base case holds since

$$\begin{aligned}
&hd(tl^0(filter(p, tl^{\alpha_{n-1}+1}(z)))) \\
&\quad \equiv hd(hd(tl^{\alpha_n}(z)) : filter(p, tl^{\alpha_n+1}(z))) && \text{Thm. 9-3} \\
&\quad \equiv hd(tl^{\alpha_n}(z)) && \text{prop. of } cons \text{ and } hd \\
&\quad \equiv x_{\alpha_n} && \text{def. of } z \\
&\quad \equiv x'_n && \text{def. of } \alpha_k.
\end{aligned}$$

The induction step holds since

$$\begin{aligned}
&hd(tl^k(filter(p, tl^{\alpha_{n-1}+1}(z)))) \\
&\quad \equiv hd(tl^k(hd(tl^{\alpha_n}(z)) : filter(p, tl^{\alpha_n+1}(z)))) && \text{Thm. 9-3} \\
&\quad \equiv hd(tl^{k-1}(filter(p, tl^{\alpha_n+1}(z)))) && \text{prop. of } cons \text{ and } tl \\
&\quad \equiv x'_{k+n} && \text{ind. hyp. } \square
\end{aligned}$$

Using *from* and *filter*, we can define a function that generates the prime numbers. This function is based on an example in [5] which is attributed to P. Quarendon.

$$\begin{aligned}
sieve(l) &\Leftarrow hd(l) : sieve(filter(hd(l), tl(l))) \\
primes &\Leftarrow sieve(from(2)).
\end{aligned}$$

We shall prove that *primes* is the list of prime numbers by showing that $hd(tl^k(primes)) \equiv p_k$ for all $k \geq 0$, where p_k is the kth prime number.

Theorem 9-5. $hd(tl^k(primes)) \equiv p_k$

Proof: Inductively define $x_{n,k}$ for all non-negative integers n and k, as follows. Let $x_{0,k} = k + 2$ for all non-negative integers k. For every non-negative integer n, let $x_{n+1,0}$ be the smallest element of $\{x_{n,j} \mid j \geq 0\}$ not divisible by $x_{n,0}$. For every pair of non-negative integers n and k, let $x_{n+1,k+1}$ be the smallest element of $\{x_{n,j} \mid j \geq 0 \wedge x_{n,j} > x_{n+1,k}\}$ not divisible by $x_{n,0}$. Elementary number theory shows that $p_n = x_{n,0}$ for all non-negative integers n. For all $n \geq 0$, let z_n be the list such that $hd(tl^k(z_n)) \equiv x_{n,k}$ for all $k \geq 0$.

By Theorem 9-4, $filter(x_{n,0}, z_n) \equiv z_{n+1}$ for all non-negative integers n. Also, notice that

$$\begin{aligned}
filter(hd(z_n), z_n) \equiv\ & \textbf{if } (hd(z_n) \textbf{ mod } hd(z_n)) = 0 && \text{def. of } filter \\
& \quad \textbf{then } filter(hd(z_n), tl(z_n)) \\
& \quad \textbf{else } hd(z_n) : filter(hd(z_n), tl(z_n)) \\
\equiv\ & filter(hd(z_n), tl(z_n)) && \text{prop. of } \textbf{mod} \text{ and } \textbf{if}.
\end{aligned}$$

Now we are ready to show by induction on k that

$$hd(tl^k(sieve(z_n))) \equiv p_{n+k}.$$

The base case holds since

$$\begin{aligned}
&hd(tl^0(sieve(z_n))) \\
&\quad \equiv hd(hd(z_n) : sieve(filter(hd(z_n), tl(z_n))) && \text{def. of } sieve \\
&\quad \equiv hd(z_n) && \text{prop. of } cons \text{ and } hd \\
&\quad \equiv x_{n,0} && \text{def. of } z_n \\
&\quad \equiv p_n && \text{previously shown.}
\end{aligned}$$

The induction step holds since

$$\begin{aligned}
&hd(tl^k(sieve(z_n))) \\
&\quad \equiv hd(tl^k(hd(z_n) : sieve(filter(hd(z_n), tl(z_n))))) && \text{def. of } sieve \\
&\quad \equiv hd(tl^{k-1}(sieve(filter(hd(z_n), tl(z_n))))) && \text{prop. of } cons \text{ and } tl \\
&\quad = hd(tl^{k-1}(sieve(filter(hd(z_n), z_n)))) && \text{previously shown} \\
&\quad \equiv hd(tl^{k-1}(sieve(filter(x_{n,0}, z_n)))) && \text{def. of } z_n \\
&\quad \equiv hd(tl^{k-1}(sieve(z_{n+1}))) && \text{Thm. 9-4} \\
&\quad \equiv p_{n+k} && \text{ind. hyp.}
\end{aligned}$$

The theorem follows easily from this result since

$$\begin{aligned}
hd(tl^k(primes)) &\equiv hd(tl^k(sieve(from(2)))) && \text{def. of } primes \\
&\equiv hd(tl^k(sieve(z_0))) && \text{Thm. 9-2 and def. of } z_0 \\
&\equiv p_k && \text{previously shown. } \square
\end{aligned}$$

10. Conclusions

We have given strict and non-strict semantics for a simple functional programming language. These semantics were specified by giving axioms for the domains and semantic functions involved. These semantics provide powerful and uniform methods for proving the correctness of programs in both the strict and the non-strict cases. The axioms for both semantics have exactly one model, up to isomorphism. We have also described an interpreter that satisfies the non-strict semantics.

The primary conclusion to be drawn from this work is that strict and non-strict semantics for lists are not as different as they might at first seem. The axiomatizations we have given for the two semantics differ only slightly in form, and the proof techniques we have demonstrated for the two semantics are quite similar. If we had given the semantics be positing a different domain for each case and defining interpretation functions over those domains, (the more conventional approach) then

many of the similarities between the strict and the non-strict semantics would have been obscured. Of course, there are differences, the existence of infinite objects in non-strict semantics perhaps being the most significant. But recognizing the similarities between strict and non-strict semantics makes it easier to fully exploit their differences.

11. Acknowledgements

Alain Martin served as my advisor for this project, his guidance and patience were essential to its completion. Young-il Choo provided many very helpful discussions, and taught me a lot about formal semantics. Members of Alain Martin's group of graduate student's at Caltech, particularly Kevin Van Horn, made several helpful suggestions for improving an earlier draft. Steve Brookes also made several suggestions, and discovered several errors.

12. References

[1] Cartwright, Robert and Donahue, James, 'The Semantics of Lazy (And Industrious) Evaluation', *1982 ACM Symposium on LISP and Functional Programming,* 253-264.

[2] Choo, Young-il, 'An Inverse Limit Construction of a Domain of Infinite Lists', 5188:TR:85, Dept. of Computer Science, California Institute of Technology, 1985.

[3] Dijkstra, E.W., *A Discipline of Programming,* Prentice-Hall, 1976.

[4] Friedman, D. P., and Wise, D. S., 'CONS Should Not Evaluate Its Arguments', *Automata, Languages, and Programming,* Edinburgh University Press, 1976, 257-284.

[5] Henderson, Peter, *Functional Programming: Application and Implementation,* Prentice-Hall, 1980.

[6] Manna, Zohar, *Mathematical Theory of Computation,* McGraw-Hill, 1984.

[7] Martin, Alain J., 'A General Proof Rule for Procedures in Predicate Transformer Semantics', *Acta Informatica* **20** (1983), 301–313.

[8] Rem, Martin and Martin, Alain J., 'Lecture Notes on Procedures', MR85/2, Dept. of Computer Science, California Institute of Technology, 1985.